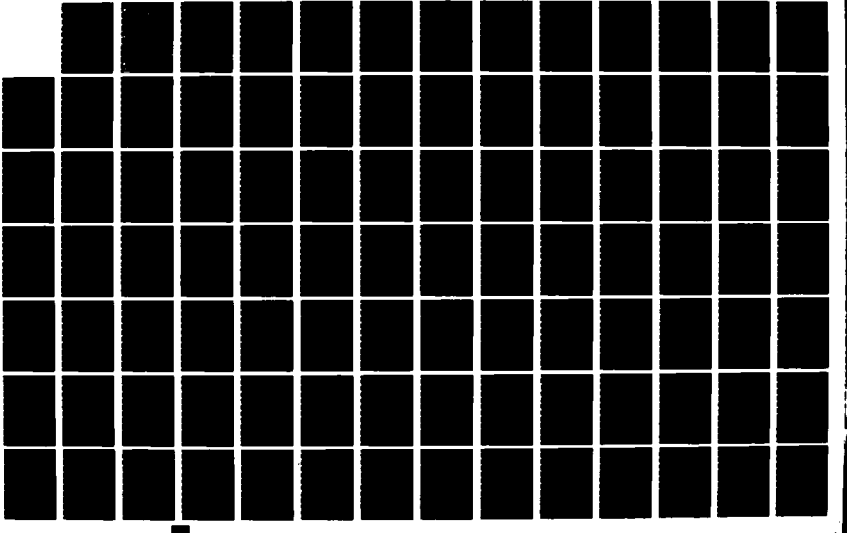
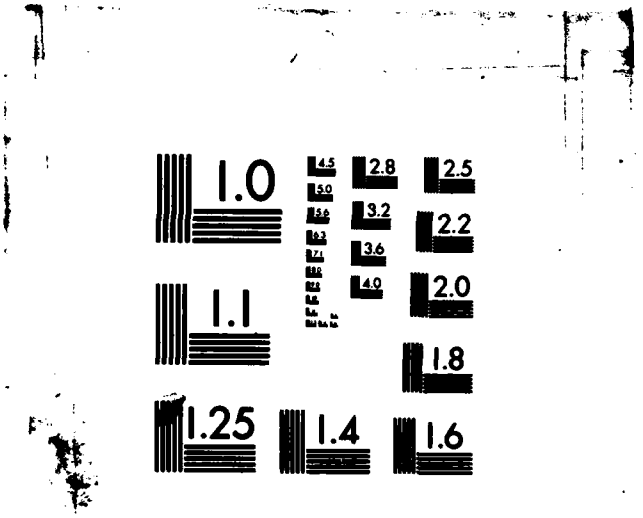
AD-A138 413 A MICROCOMPUTER BASED SYSTEM FOR ANALYSIS OF LINE DRAWING QUANTIZATION TECHNIQUES(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. J E ROCK 1/2

UNCLASSIFIED 09 DEC 83 AFIT/GE/EE/83D-77 F/G 9/2 NL





A MICROCOMPUTER BASED SYSTEM FOR ANALYSIS OF LINE DRAWING QUANTIZATION TECHNIQUES

THESIS

AFIT/GE/EE/83D-77 Joseph E. Rock, Jr.
1Lt. USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

84 02 29 053

AFIT/GE/EE/83D-77

A MICROCOMPUTER BASED SYSTEM FOR ANALYSIS OF LINE DRAWING QUANTIZATION TECHNIQUES

THESIS

AFIT/GE/EE/83D-77 Joseph E. Rock, Jr.
1Lt. USAF

AFIT/GE/EE/83D-77

A MICROCOMPUTER BASED SYSTEM FOR ANALYSIS OF LINE DRAWING QUANTIZATION TECHNIQUES

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by
Joseph E. Rock, Jr., B.S.
1st Lt. USAF
Graduate Electrical Engineering
December 1983

A-1

Acknowledgments

I would like to thank my advisor, Major Ken Castor of the Air Force Institute of Technology, for proposing this thesis topic. I am very grateful for his assistance and support throughout the project. I would like to thank my friends for their constant support and encouragement.

Contents

List of Figures

Abstract

This paper documents the design and implementation of a system for analysis of line drawing quantization techniques on a microcomputer system. The system provides software tools for the input, output, and analysis of line drawings and their quantized representations. The system uses a digitized version of the original line drawing as the basis for all performance calculations.

A program to compute the single ring chain code of the input digitized line drawing was implemented. The performance of the chain coded versions of the line drawings could then be calculated. This project developed all the necessary tools for the analysis of any line drawing quantization technique that can be implemented on a microcomputer system.

I. Introduction

There are many applications in which the ability to store and analyze two-dimensional images efficiently is very important. The most popular way to represent a two-dimensional image is by two-dimensional sampling (simply a set of x-y coordinates each with an appropriate value indicating the level of the image at that point). The major drawback of this method is that the amount of data, and therefore the requirement for memory and processing speed, increases drastically as the resolution of the image increases. Two-dimensional sampling is necessary for some applications, but there are other applications that are more efficiently done using a different method.

One such application is the line drawing. A line drawing is an image which consists entirely of thin lines on a contrasting background. Examples of line drawings are contour maps, graphs, and simple writing. A set of schemes known as generalized chain codes has been developed to more efficiently represent line drawings [1,8]. Currently there is no theoretical way to measure the performance of these codes in a quantitative manner.

The objective of this thesis is to develop a quantitative measure of the performance of a subset of generalized chain codes when they are used to represent an arbitrary

line drawing. The line drawing will be input using a digitizer and the digitized version of the line then used as if it were the actual line drawing. The chain code quantization of the line drawing is then done off-line using the digitized line drawing as the basis for the quantization. This quantized image was then compared to the digitized line drawing to compute performance measures.

The entire development and analysis of the problem as described in the preceding paragraph was to be done on a Heathkit H89 microcomputer system. The system configuration as I began this thesis was as follows:

1.) Heathkit H89 microcomputer with 1 serial I/O port, one 90k byte disk drive and two 594k byte disk drives

2.) Heathkit H25 printer already interfaced to the microcomputer

3.) Hewlett-Packard model 9874A digitizer and an ICS Electronics Corporation model 4885A IEEE-488 bus controller with RS-232 interface

4.) Houston Instruments model DMP7 x-y plotter with a RS-232 serial interface

The digitizer and the plotter had to be interfaced to the H89 and software written to drive these two devices.

The organization of this thesis follows the approach to the problem. In Chapter 2, the work done to integrate all of the devices into a working system and the software to interface these devices to a higher level language (Pascal)

are described. Chapter 3 is a description of the main graphical input and output programs and the data file format used throughout this project. Chapter 4 contains a detailed description of chain codes in general as well as the specific chain codes and performance measures used in this thesis. This is followed in Chapter 5 by a description of the actual implementation of the chain codes and the performance calculation algorithms. Chapter 6 contains the summary and recomendations.

II. System Integration

The first problem that had to be solved when I began work on this thesis was to interface the system components . The equipment that was used for this project included a Heathkit H89 microcomputer, a Heathkit H25 printer, a Hewlett-Packard model 9874A digitizer, an ICS Electronics Corporation model 4885A bus controller to control an IEEE-488 bus via an RS-232 serial port, and a Houston Instrument model DMP7 digital plotter with RS-232 serial interface. The microcomputer itself has one 90k byte and two 594k byte disk drives and the printer was already interfaced to the microcomputer. To interface the digitizer and the plotter it was necessary to add two RS-232 serial ports to the H89 microcomputer. This item was easily done once the expansion kit from Heathkit was purchased. Getting the digitizer and the plotter to work under program control was the difficult part.

This chapter will discuss the interfacing of the digitizer to the microcomputer and then the interfacing of the plotter to the microcomputer. When both of these items were interfaced, the idea was to enable them to be controlled from a higher level language, in this case Pascal, so the lower level routines were developed to accommodate this thinking. The microcomputer operating system is CP/M and

the Pascal that was used is Digital Research's Pascal MT+. All of the low level routines were written so that it is possible to exit from a routine that is waiting for input from a device simply by typing a CNTRL-C on the keyboard. The routines indicate the abnormal exit by using an error flag which the higher level routines then have the responsibility to check. All the software incorporated this feature so that it is easy to recover from any abnormal condition without having to reboot. Listings of the actual assembler code and the Pascal interfacing routines developed can be found in Appendix A.

Using the routines developed to interface the digitizer and plotter to the microcomputer, the only hardware specification that the user needs to be aware of is that the digitizer has a resolution of 0.001 inches and the plotter has a resolution of 0.005 inches. Both of these devices communicate with ASCII characters and use only integer numbers (i.e. they communicate in integer multiples of their resolution).

Interfacing the Digitizer

To input the line drawings into the microcomputer a Hewlett-Packard (HP) model 9874A digitizer that communicated with its host via a HPIB (IEEE-488) bus was used. An ICS Electronics Corporation model 4885A IEEE-488 bus controller with an RS-232 serial interface was required to drive the digitizer from the microcomputer. This bus controller was necessary because the microcomputer has only RS-232 serial

ports. The first problem to solve was to establish the hardware connection. After this was done, the assembler routines to interface the IEEE-488 bus controller to the microcomputer were developed. These routines were then used as subroutines for the Pascal interface routines. These Pascal routines make it possible to interface to the digitizer at a reasonably high level.

Initially, the microcomputer had only one serial port and that was used for the printer so the first time any commands were sent to digitizer the sending device was a simple ASCII terminal. By using this method of utilizing a simple terminal as the host, the hardware connections were verified independant of the development of the assembler routines on the microcomputer. The hardware connection consisted of constructing an RS-232 cable to connect the bus controller to the microcomputer.

The software to interface the digitizer to a high level language, Pascal, was developed in three basic steps. First, an assembly language program that made the microcomputer emulate a simple terminal was developed. I then used the low level routines of that program as the basis for the low level routines that would eventually become the standard interface routines. These interface routines were then used as subroutines for the Pascal digitizer interface routines.

The low level routines that were written include:

1.) BUSINT: a serial port and bus controller initialization routine

2.) BUSIN: an IEEE-488 bus device input routine

3.) BUSOUT: an routine to output to a IEEE-488 bus device

All of these routines are written in Z80 assembler code and the source for the routines is in BUSLIB.MAC. These routines were written with the use of Digital Research's M80 assembler in mind. This was done because M80 relocatable code (.ERL) files can be easily linked to Pascal MT+ relocatable code (.REL) files. To link the files the ERL file is renamed with the REL extent and linked using the Pascal MT+ linker. BUSOUT and BUSIN transfer a group of characters terminated by a carriage return (CR) to or from the specified IEEE-488 bus device. These routines effectively make the bus controller transparent to the calling program. The specific information for calling these routines is contained in the source code.

Using the basic interface routines to do simple input/output from/to the digitizer, a set of routines were developed that would enable a calling program to get the data for a point from the digitizer by calling one routine. This set of routines was written in Pascal and the source is in DIGRTNS.SRC. To get a point from the digitizer the calling routine calls GETPOINT and GETPOINT will return with one of the following:

1.) If there is no special function key entered and no digitizer error then the coordinates of the digitized point, the pen up/down indicator, and the annotation number

entered from the digitizer keypad are returned.

2.) If a special function key except Fa is entered then the number of the special function key that was entered is returned (the Fa key allows the user to enter the parameters for continuous sampling from the H89) (a function key number not equal to 0 indicates that a function key was entered).

3.) If the digitizer indicates an error other than an out-of-bounds error then a true value for the error condition indicator and a message on the screen of the H89 are returned (the out-of-bounds error is handled by the GETPOINT routine).

With these routines all that the main program has to do is initialize the bus controller (call BUSINT) and then start getting points with GETPOINT. The main program can have the special function keys (except Fa) represent whatever functions it wants.

Interfacing the Plotter

Interfacing the plotter to the microcomputer was an easier task than interfacing the digitizer to the microcomputer. This is because the plotter has an RS-232 interface as its standard interface. The first task was to make a cable to connect the plotter to the microcomputer. The next step was to develop the interface routines (assembler code) which would interface to a Pascal program.

The assembly language routines that I developed are:

1.) PORTIN: a routine to initialize the serial port

2.) CHAROT: a routine to output a single character

to the plotter

3.) CHARIN: a routine to input a single character from the plotter (mainly used to handshake with the plotter)

4.) LINOUT: a routine to output a group of characters terminated by a '}' to the plotter

These were the only plotter specific routines that were written because the most efficient set of higher level routines depend heavily on the specific application. The set of commands that the plotter has is also at a high enough level to enable plotter routines to be developed very easily. A summary of the plotter commands is given in the next chapter.

Summary

This chapter described the interfacing of the digitizer and the plotter to the microcomputer. This interfacing included a brief description of the hardware and a slightly more detailed description of the software. The software routines were described by a brief functional description of the user callable routines.

III. Graphical Data Input/Output and Storage

The purpose of this chapter is to describe the graphical data input and output programs and the format of the data when it is stored on a floppy disk. First, the disk file format is described and then the main programs used to input data from the digitizer and output data to the plotter are described. Programs to have the plotter do lettering or simple drawing from the microcomputer, and a program to have digitized data echoed directly to the plotter are also described. The programs to input data from the digitizer and to output data to the plotter are the basic input/output tools for this project. Complete Pascal source listings for the programs described in this chapter can be found in Appendix B.

Disk File Format

To make it possible to have to write only one file output program and to make it easier to do the performance calculations, a standard file format was defined which was then used for the storage of all sets of points. A line in a point file contains a pen up/down indicator and the x and y values of the point as its first three items. The format of a line after the three items is then determined by the program that generated it (i.e. available for free format until end of line). The pen indicator and the x and y

values are all that is necessary to describe any line segment. There must be at least one space on each side of a number that is to be read in and each line must be terminated with a carriage return.

To properly describe a line, the first point(s) of a file must have the pen up to indicate where the line begins. If there are no pen up lines then it is impossible to consistently determine where the line begins. This problem does not exist at the end of a line because a line segment can be terminated with a pen up or an end of file condition. Also, the lack of a pen-up line at the beginning of a file will cause the performance calculation program to generate incorrect results (it then assumes the drawing began at (0,0)). A file may contain any number of separate line segments separated by pen-up lines.

The pen position indicator is in the first column of every line of the file. The pen position indicator itself is simply a single character: 'U' for pen up and 'D' for pen down. The order of describing the line is that the pen is put up/down and then moved to the location indicated by the x and y coordinates on the line. The x and y coordinates are always integer numbers with a magnitude of less than 32,765 but are otherwise not restricted. Normally the unit that is used in the files is that 1 represents 0.001 inches. This is due to the fact that the resolution of the digitizer is 0.001 inches. Also, for the performance computations to be done correctly the unit of length used in both input

files must be the same.

To insure that the pen position indicator was always correct without having to worry about timing, a decision was made to make the person doing the digitizing manually indicate when to put the pen up and when to put the pen down. This indication is done by using the special function keys on the digitizer.

The format of the line after the first three items is not standard. For a file of points input from the digitizer a line contains (after U/D x y) the pen position indicator output by the digitizer and the annotation number that was entered from the digitizer keypad. A file of points generated by the chain coding program contains (after U/D x y) the chain code and also contains the gridsize and level of the code if the line is the first of a new line segment.

Digitizer Input Program

The digitizer input program (DIGITIZE) allows the user to create a file(s) of data points that can then be used for whatever purpose that the user generated them. The program is very easy to use because the program prompts the user for all the input it needs. The user must enter the filename for the digitized data and then set the sampling mode of the digitizer. After that is done the user simply takes points until all of the data for a line drawing has been collected. Once the user has entered the initial parameters, all input is initiated from the digitizer either by the cursor or the special function keys.

The definition of the special function keys on the

digitizer is as follows:

Fa: set sampling parameters

Fb: indicate end of data file and end program or start a new file of data

Fc: indicate pen down

Fd through prefix Fe: indicate pen up

When Fa or Fb are entered the user must make additional input from the microcomputer. Again, all input is prompted and, where appropriate, defaults are provided.

Plotter Output Program

The plotter output program (PLOTFILE) allows the user to plot any file of points which adhere to the format described earlier in this chapter. The program prompts the user for all necessary inputs and allows the user to scale and translate the line drawing for plotting. The user must enter the filename of the data points, select the line type for the plotter to use, enter the scaling factor, and then enter the x and y translations. The plotter output program will then plot the entire file using handshaking with the plotter.

The scaling factor is applied by multiplying the x and y values by the scaling factor. After the values are scaled, the specified translation factors are added to the values. The values are then sent to the plotter. The scaling factor can be any real number but the translation factors must be integers (all factors must have a magnitude of less than 32,765). One thing to consider for the trans-

lation and scaling factors is that 200 equals one inch on the plotter and 1000 equals one inch on the digitizer.

The line type is an integer number (0 - 8) and the available line types are shown in Figure 3-1 [4].

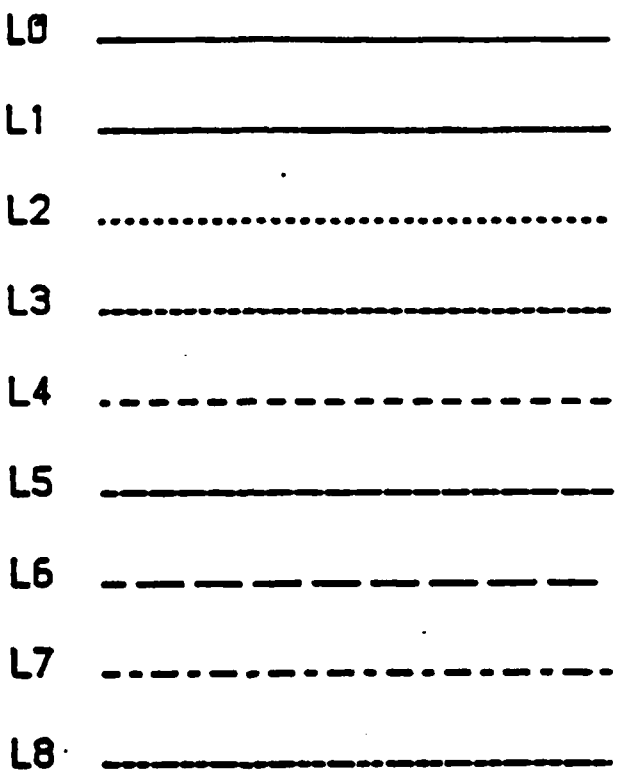


Figure 3-1 Plotter Line Types

One effective way of showing the difference in the original line and the coded line is to place the two lines directly on top of each other but use a different line type or color of the line (pens must be changed manually).

Direct Plotter Control Program

The program that provides direct plotter control (LABELS) is not at as high a level as the two programs described previously. This was done to maximize the program's flexibility. All this program provides is a way to

send commands directly to the plotter which are executed after the pen has been placed at the position input by the user. The user must use the command set and format that the plotter uses for normal input. This command set is very easy to use. The program also provides the option to repeat a command string any number of times after changing the starting position by user specified increments in the x and y directions. The program does not currently support back-space correcting of command strings. To avoid sending an incorrect command to the plotter, specify a repetition number of zero.

A summary of the plotter command set and the necessary syntax follows.

O - set origin: the current pen location becomes the new origin

D - pen down: puts the pen down at the current location

U - pen up: immediately picks the pen up

H - home: moves the pen to the home location (lower left corner) and defines that location as the new origin

A - absolute plotting mode: all coordinates will be plotted with respect to the currently defined origin

R - relative plotting mode: all coordinates will be plotted with respect to the point plotted immediately prior to the point being currently plotted

Ln - set line type: define line type as n (see line type definition above)

Srhbsss_ - symbol plotting: plot ASCII character string sss with rotation r and height h ('_' indicates end

of character string, b is a space) (r is an integer 1 - 4 and the rotation is: rotation = (r-1)*90 degrees, 0 degrees is straight right and the rotation angle is positive clockwise) (h is an integer 1 - 5 which corresponds to hieghts of: 1 = 0.07", 2 = 0.14", 3 = 0.28", 4 = 0.56", and 5 = 1.12")

T - self-test routine: perform self-test program

x,y - move pen to x,y: move the pen to x,y with respect to the origin or previous point (A vs R) with the pen up or down (U vs D) (x and y are integers sent as ASCII character strings)

As can be seen by the set of commands that the plotter has, it is not very difficult to do simple labeling or create very simple one-time drawings using the LABELS program. The program was not designed to do very complex labeling or drawings and should not be used for them. A modification of this program where it would get its input from a disk file would be reccomended for repetitious or complex drawings.

Digitizer to Plotter Program

One program (DIGPLOT) was designed that combined the functions of DIGITIZE and PLOTFILE but eliminated the use of a disk file. This program echoes the digitized data points directly to the plotter after scaling and translating them. The program prompts the user for all inputs and the program is exited by typing a CNTRL-C on the keyboard. After the program is initialized the special function keys are the same as for DIGITIZE except that Fb now allows the user to

redefine the plotter parameters (line type, scaling, and translation). The major use of this program is as a demonstration program.

Summary

This chapter described the main input and output programs to handle the digitized point data and the standard file format for such data. The descriptions provided are thorough enough to get the user started on the system. All of the programs are user friendly enough to prompt the user through the process of inputting and outputting data.

IV. Chain Codes and Performance Measures

The chain code is a method of quantizing and encoding line drawing images. The quantization is done by superimposing a grid of some specified gridsize onto the line drawing and selecting a set of nodes to represent the drawing. A node is defined as the intersection of the horizontal and vertical grid lines. A line segment is described by a sequence of nodes originating at the first point of the line segment and terminating at the last point of the line segment. A node is only identified in relation to the node which immediately preceded it in the sequence, hence the name chain code.

All chain codes follow this basic structure. The various forms of the code differ in the rules which govern the selection of the nodes which represent the line drawing and the set of symbols used to encode the nodes. This chapter continues with a more detailed description of some of the forms of chain coding and specifically the form of the chain code that was used in this project. Following this description will be an explanation of the theoretical application that was assumed and the coding algorithm that was used.

The overall objective of this thesis was to obtain a way to quantitatively evaluate the performance of various codes on different line drawings. The performance measure

that was used is based on the surface area between the line drawing and the coded version of that line drawing and the length of line. The digitized line drawing is assumed to be equivalent to the actual line drawing. This assumption can be made because the smallest gridsize to be used is at least ten times the resolution of the digitizer. A complete discussion of this performance measure and the alogrithm that was used to implement it follows the chain code description.

The Grid Intersect Code

The grid intersect code is conceptually one of the two simplest forms of the chain code. For any chain code you must be given, or have assumed, some starting point. The starting point must be a node. This first node may be the node closest to the first grid intersection or it may be the first point of the line drawing. When the first of these methods is used the grid is a fixed overlay on the line drawing. It is usually fixed with respect to some origin. When the second of these two methods is used the first point of the line drawing is the location of the first node and the grid is layed out with respect to this first node. The set of rules governing the selection of the next node for the grid intersect code are [1]:

1.) Trace the line from the current node until it intersects any grid line.

2.) Select the node which is closest to the grid intersect.

3.) If this node is not the current node then add the

code for the node to the chain and make this node the current node.

4.) If this node is the current node then continue with the trace (step 1) from the current intersection (the same node is never specified twice in a row).

5.) If the line drawing has not ended then repeat the process starting with 1.

The current node and the possible next nodes are defined by the drawing in Figure 4-1.

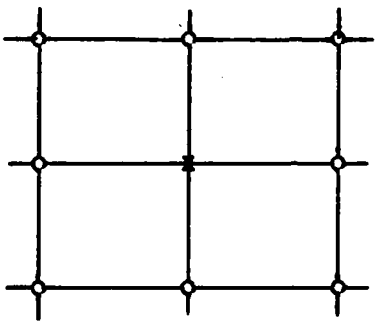


Figure 4-1 Possible Next Nodes of Ring 1

The eight nodes which make up the square ring of next possible nodes is called ring 1 or a single ring of level 1. Considering this one ring as the basis of a code, the next variation of the chain code is formed. This form of the chain code will be called the single ring chain code of level 1.

For the single ring chain code, the rules governing the selection of the next node are as follows:

1.) Trace the line until it intersects the single ring.

2.) Determine the node closest to the intersection.

3.) Add the code for that node to the chain and make

that node the current node.

4.) If the line has not ended then repeat the process In general, the single ring chain code will have fewer nodes in the chain and be not quite as accurate as the grid intersect code.

The encoding procedure for a chain code is simply a method of assigning symbols to the nodes used in the code. The assignment of these symbols is arbitrary but must be consistent in the encoding and decoding process. For the grid intersect code and the single ring code of level 1 there are eight possible nodes. The method of assignment used in this project is shown in Figure 4-2.

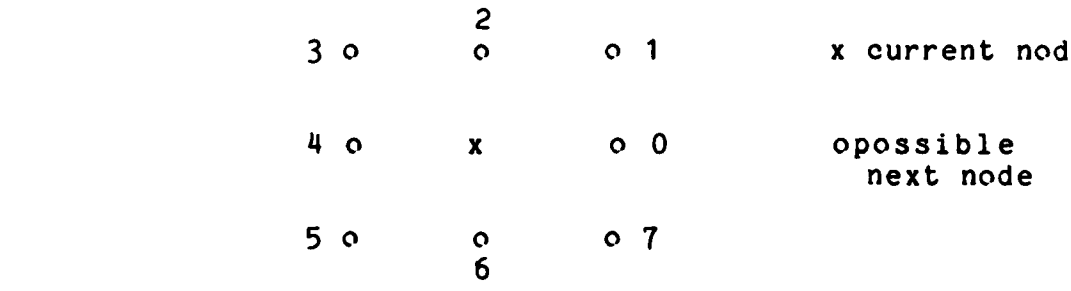
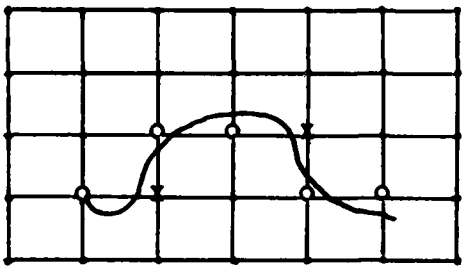


Figure 4-2 Encoding Assignment for Ring 1

For this ring assignment, every node in the grid intersect code or the single ring chain code of level 1 would be represented by three bits (000 through 111). An example of the use of the grid intersect code and the single ring chain code of level 1 is shown in Figure 4-3. This example also demonstrates the difference between these two codes.



o - chosen by both codes

x - chosen by grid intersect only

Grid intersect code: 020060
Single ring code : 1070

Figure 4-3 Example of Ring 1 Codes

Extensions of the Single Ring Code

The single ring code can easily be extended to levels greater than one. A single ring code of level 2 would have 16 possible next nodes as shown in Figure 4-4.

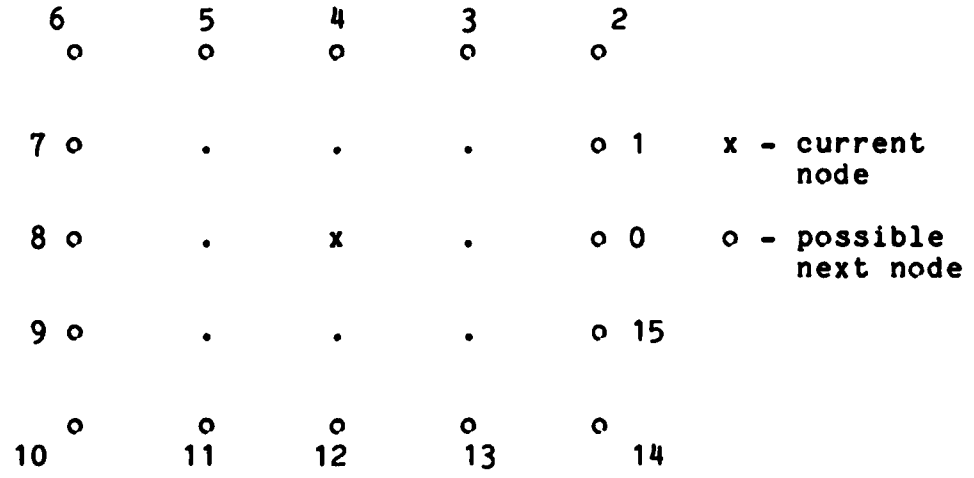


Figure 4-4 Nodes of Level 2 Single Ring

The single ring code of level n uses the same rules of coding as the single ring code of level 1 except that the ring that is intersected is the level n single ring. The advantage of increasing the level of the code is that the angular resolution of the code is better. The disadvantage

is that the code with level 2 requires four bits per node as opposed to three bits per node. The gridsize of the code is still defined as the distance between grid lines and therefore the line being coded must move 2*Gridsize in the x or y direction before a node is coded.

Extending the single ring code even farther than the second level, it can be shown that the number of possible next nodes is 8*Level. The angular resolution of the code does not double with consecutively higher levels. The chain coding program developed for this thesis generates the single ring chain code of any level and gridsize that is specified for a given line drawing.

Higher Order Codes

Higher order codes can now by formed by using a combination of two or more single rings of different levels. Higher order codes of this type are also known as adaptive chain codes. The reason for this is that where the radius of curvature of the line drawing is large the higher level rings are used and where the radius of curvature is small the lower level ring(s) is used. In this way the accuracy of coded version of line is maintained while the number of nodes in a nearly straight section of the line is kept to a minimum. Chain codes of this type will be identified by the rings used in the code. A chain code using level 1 and level 3 single rings would be called a single ring of level (1,3).

The rules governing the selection of next nodes of a

single ring chain code of level (i,j,k) are: (i < j < k)

1.) Trace the line until it intersects the level k single ring.

2.) If the line stays within the area of precision of the node closest to the point where it intersected the ring, then add that node to the chain, make it the current node, and then go to step 1.

3.) If the line stays within the area of precision of the node closest to the point where it intersected the level j single ring then add that node to the chain, make it the current node, and then go to step 1.

4.) If the line fails both 2 and 3 then select the node on the level i single ring according to the coding method used for the lowest level ring. The method used may be the method for a single ring code of level i or for the grid intersect code of level i. The appropriate node is added to the chain and made the current node and then the process is repeated.

These four steps are repeated until the end of the line is encountered. Implied in these steps is the ability to store the segment of the line from the current node to where it intersects the highest level ring used in the code. The area of precision of a node is the area enclosed by two lines originating at the current node and terminating at the midpoints of the line segments between the node and its two adjacent nodes on the same ring and the node's ring. An example of the area of precision of a node on a level 3 ring is shown in Figure 4-5.

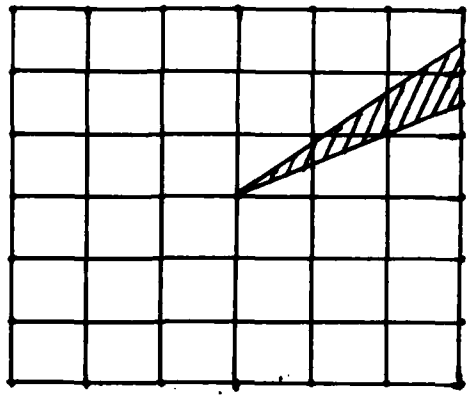


Figure 4-5 Area of Precision of a Node on the Level 3 Ring

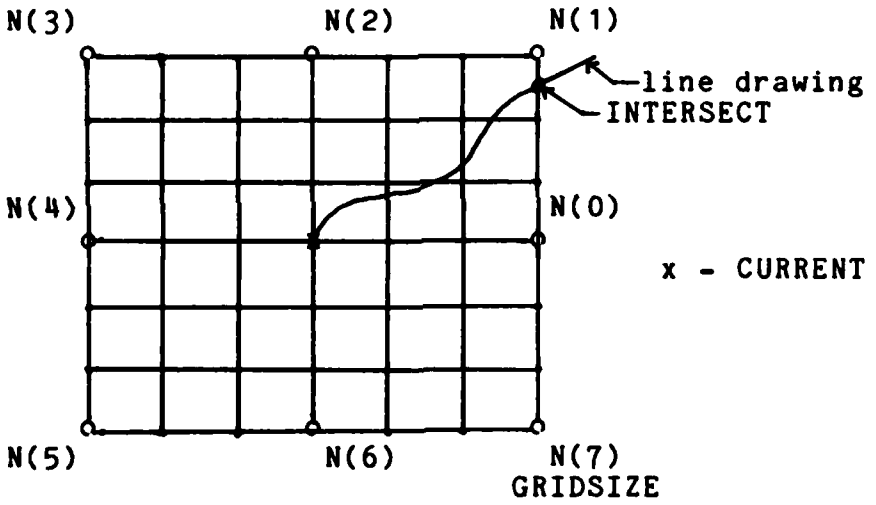
The encoding procedure for nodes of the higher level codes is similiar to the encoding procedure for the simple single ring codes. The encoding and decoding procedures must be the same and every node used in the code must have a unique symbol to identify it. Other than this restriction, the assignment of symbols to nodes is arbitrary.

Theoretical Implementation

The method of chain coding implemented in this thesis is the single ring chain code of level n. The first point of each line segment in a line drawing is used as the origin of the grid. Each continuous chain is started with the coordinates of the first point of the line segment and terminated with the last point of the line segment. Using this implementation, the minimum that is needed to represent a line drawing is the coordinates of the first point of the line segment, the codes of the nodes in the chain until the end of the continuous line segment is reached, a code to indicate the end of the line segment, and the coordinates

of the last point of the line segment. All subsequent line segments in the line drawing would be represented by additional sets of these four elements. After the entire line drawing is done a unique code is used to indicate the end of the line drawing.

In this thesis, the codes of the nodes are all positive integers, the code to indicate the end of a line segment is a -1, and the end of line drawing code is the end of file indicator. The algorithm used to compute the chain code is described next (refer to Figure 4-6 for variable definitions).



for this example LEVEL = 3

(POINT(X),POINT(Y)) are coordinates along line drawing
(XCODE,YCODE) = coordinates of the selected node in integer multiples of the gridsize where (0,0) is the lower left corner of the single ring
CODE = code number of selected node
DELTAX = Absolute[POINT(X) - CURRENT(X)]
DELTAY = Absolute[POINT(Y) - CURRENT(Y)]

The nodes N(i) will always have code numbers such that

code(N(i)) = i * LEVEL

Figure 4-6 Variable Definitions for Chain Coding Algorithm

0.) add coordinates of first point of first line segment to chain

1.) trace line until (DELTAX or DELTAY is greater than or equal to LEVEL * GRIDSIZE) or end of line segment encountered

2.) if end of line segment then go to step 10.

3.) compute the x and y values of INTERSECT

4.) compute
DELTAINTER = (LEVEL * GRIDSIZE)+INTERSECT-CURRENT

5.) compute
XCODE = ROUND[DELTAINTER(X) / GRIDSIZE]
YCODE = ROUND[DELTAINTER(Y) / GRIDSIZE]

6.) compute
CODE = XCODE + YCODE
(CODE = 0 is now lower left)

7.) if XCODE < YCODE then CODE = LEVEL * 8 - CODE
(make CODE monotonic counter-clockwise)

8.) if CODE < (3 * LEVEL)
then CODE = CODE + LEVEL * 5
else CODE = CODE - LEVEL * 3
(translate CODE such that CODE = 0 will be straight right from current node)

9.) add code to chain and make the new node the current node and then go to step 1.

10.) CODE = -1, add coordinates of last point to code

11.) if not end of line drawing
then add coordinates of first point of next line segment to code and go to step 1.
else processing is complete

The fact that this coding procedure is good for any level code is due to the behavior of the code numbers for the single ring code. The code numbers of the level n single ring nodes corresponding to the nodes of a single ring of level 1 have the following property:

code number(level n node) = code number(corresponding level 1 node) * n

This property makes the computation of the code relatively easy. A description of how this algorithm was implemented in a program is given in the following chapter.

Performance Measures

Five criteria for the evaluation of the usefulness of a general chain code have been suggested [1]: (1) compactness, (2) precision, (3) smoothness, (4) ease of encoding and decoding, and (5) facility for processing. The intended application has a significant impact on the relative importance of each of these criteria. The following is a brief discussion of the application versus the criteria and then a description of the performance measure that was used in this thesis.

Compactness is important if the primary purpose is storage or transmission. In this application the curvature characteristics of the line drawing are very important when evaluating the trade-offs between the lower-order and the higher-order codes. The lower order codes have fewer possible next nodes and therefore fewer bits per node. The higher order codes have a higher angular resolution and therefore can have a larger total gridsize and fewer total nodes.

If the accuracy with which the coded data represents the line drawing is important then the precision of the code is foremost. As with compactness, the curvature characteristics affect the trade-off but in general the smaller the grid size the better the precision of the code. Smoothness

will normally have a great deal of correlation with the accuracy but the better angular resolution of the higher order codes may give them an advantage.

The ease of encoding and decoding and the facility of processing are only important when large amounts of data are being processed. The importance of these factors is not very high when considering the high speed of digital computers. These factors will mainly be driven by the algorithm used to do the processing.

The performance measure that was designed for in this thesis is a plot of bits/unit length versus area error/unit length. The bits/unit length is a compactness measure and is defined as the total number of bits that were used in the coded representation of the line divided by the total length of the line. The area error/unit length is a precision measure and is defined as the total area enclosed between the coded represntation of the line and the line drawing divided by the total length of the line. The theoretical implementation is such that the area between the coded line and the line drawing is completely enclosed.

Performance Measure Implementation

To compute the area error/unit length, the first thing that must be done is to compute the total area error and the length of the line. The length of the line is simply the summation of the distances between consecutive points on the line. The distance between consecutive points is simply the square root of the sum of the squares of the x and y deltas. The computation of the area error is more complex and this

section will concentrate on it.

It is given that the first point and the last point of both the coded line and the digitized input line drawing are the same. From this we can see that the overlay of the coded line onto the input line drawing can be divided into a series of connected closed figures. This principle is demonstrated in Figure 4-7.

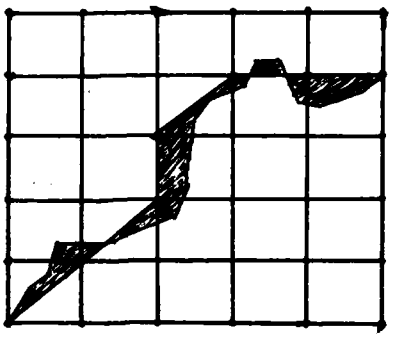


Figure 4-7 Overlay of Coded Line onto Input Line

The total area error is then the summation of the areas enclosed by all of the individual closed figures. These figures will be completely described by a series of x-y coordinates. There is a very elegant way of computing the area enclosed by a closed figure [7]. This method is restricted to closed figures that are entirely described by a sequence of x and y coordinates. Further, the sequence of coordinates must describe only one closed loop. Examples of correct and incorrect closed figures can be found in Figures 4-8 and 4-9 respectively.

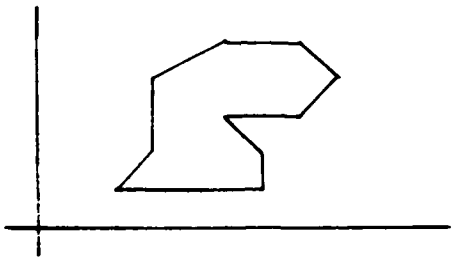


Figure 4-8 Example of Correct Closed Loop

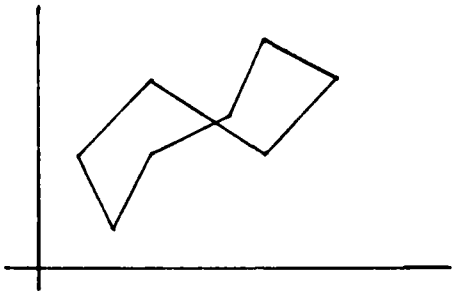


Figure 4-9 Example of Incorrect Closed Loop

The first point in the sequence describing the closed figure can be arbitrarily chosen. The algorithm to compute the area enclosed by a correct closed loop works by summing the area of trapezoids described by a consecutive pair of points on the closed figure and some reference line, in this case the x-axis. These areas will make either a positive or negative contribution to the total area of the closed figure. The sign of the individual area is determined by the relative position of the two points on the closed figure. We will assume a system for labeling the points on a closed figure as shown in Figure 4-10.

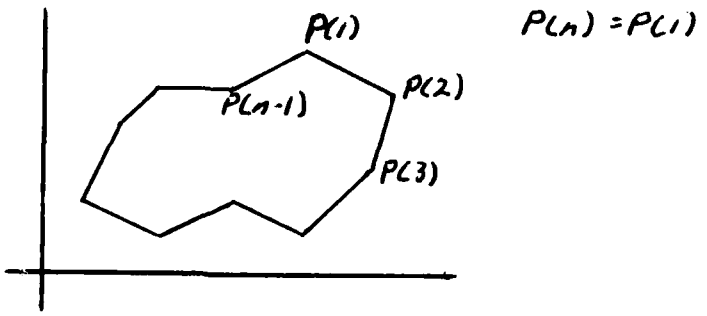


Figure 4-10 Labeling of a Closed Figure

Every trapezoid where P(i,x) is less than P(i+1,x) will make a positive contribution to the area of the closed figure. If P(i,x) is less than P(i+1,x) the contribution is negative and if P(i,x) is equal to P(i+1,x) there is no contribution. To find the total area of the closed figure, these individual area are summed around the entire closed figure (i.e. from i = 1 to n-1).

The area of the individual trapezoids is simply the average height of the trapezoid ((P(i,y) + P(i+1,y))/2) times the width of the trapezoid (!P(i,x) - P(i+1,x)!). The sign of the difference between P(i,x) and P(i+1,x) indicates whether that section will make a positive or negative contribution to the area of the closed figure. Therefore, the total area of the closed figure can be written as:

Area = SUM(i=1,n-1){[P(i,x)-P(i+1,x)] * [P(i,y)+P(i+1,y)]/2}

Now the observation can be made that if the points describing the closed figure were labeled counter-clockwise instead of clockwise then the area computed by above equation would be of the correct magnitude but be negative. To make the equation more general we take the absolute value of the computed area before adding it to the total area error.

These areas are then summed for all closed loop figures on the overlay to find the total area error. A graphical demonstration of this method on a simple closed figure is shown in Figure 4-11.

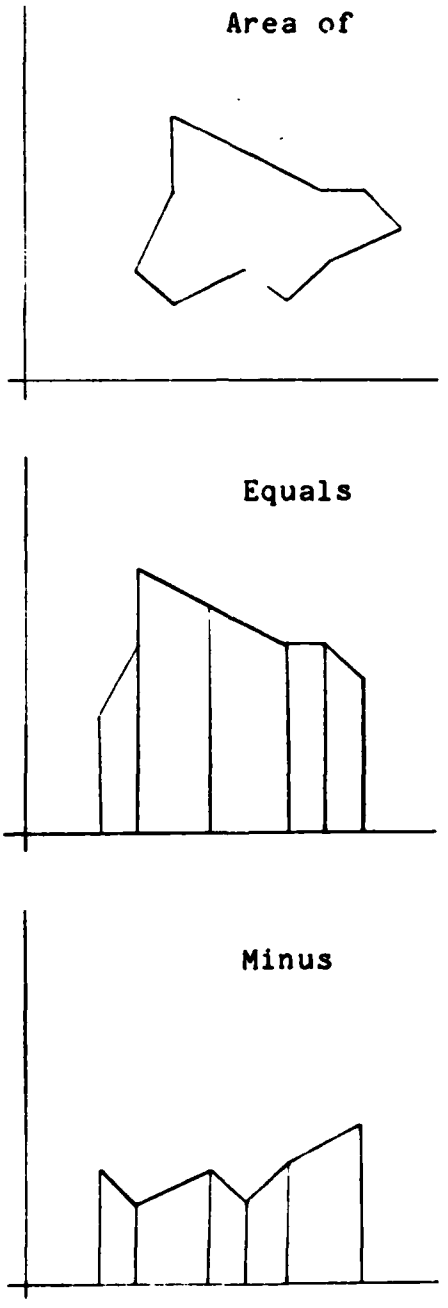


Figure 4-11 Graphical Example of Area Calculation

This method of area error calculation is in no way dependent on how the two lines were obtained. Due to this fact, any future coding programs developed for this system can use this same error computation algorithm. A description of the implementation of this algorithm can be found in the next chapter.

Summary

This chapter discussed chain codes in general and then specified the form of the chain code that was implemented in this thesis. The discussion then centered on the performance measures that can be used to evaluate the various forms of chain codes. The alogorithms used to do the coding and to compute the area error were presented in the discussion as well.

V Implementation of Chain Codes and Performance Measures

The purpose of this chapter is to describe the implementation of the single-ring chain codes and the area error per unit length calculation. The two programs to do this are CODER and ERROR respectively. Both of these programs use the file format described in chapter III. First, the single-ring chain code program will be described and then the area error per unit length program will be discussed. The source listings for both of these programs can be found in appendix C.

Implementation of CODER

The single-ring chain code program was the easier of the two to write. The program reads the points of the line to be coded from a user specified file and generates a file containing the chain codes and the coordinates of the nodes. The output file contains the coordinates of the nodes so that the standard plotting program (PLOTFILE) can be used to output the file. The chain coding is done for the level and gridsize specified by the user.

The overall operation of the program can be described by the set of steps shown below.

1.) Get parameters from user (filename of input and ouput and gridsize and level of chain code) and initialize variables.

2.) Find the first point of the very next line segment, make it the first node of that line segment, and write the coordinates out to the output file (along with a chain code of -1, the level, and the gridsize).

3.) Input points from the input file until the line drawing crosses the specified single-ring and then calculate the coordinates of the intersection.

4.) Determine the node closest to the intersection and compute the chain code for that node.

5.) Output the coordinates of this new current node and the chain code to the output file.

6.) If the end of the current line segment has not been reached then go to step 3.

7.) If the end of the line segment has been reached then output the coordinates of the end point and a chain code of -1.

8.) If the end of the line drawing has not been reached then go to step 2, otherwise close the output file and end.

The algorithm used to compute the chain code is exactly the one described in chapter IV and the details of the actual program code can be found in the source listings (they are well documented). The program prompts the user for all of the input it requires from the user.

Examples of two line drawings and the chain coded versions of the same line drawings are shown in Figures 5-1 and 5-2 respectively.

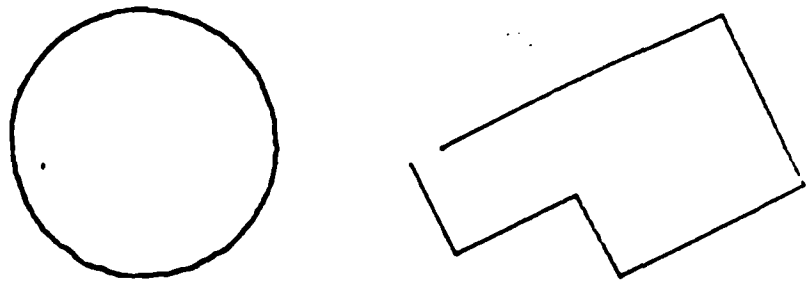
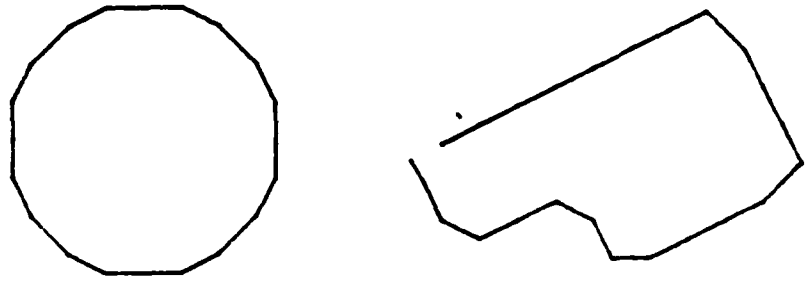


Figure 5-1 Example of a Line Drawing



Gridsize = 0.5 in Level = 2

Figure 5-2 Chain Coded Version of Line Drawing in Figure 5-1

Implementation of ERROR Program

The implementation of the area error per unit length calculation was much harder than the chain coding computation. The ERROR program has two major subroutines. These subroutines are CLOSELOOP and GROUND. The first subroutine traces along the line drawing and the coded version of the line drawing until it finds an intersection. It then computes the coordinates of this intersection and returns to the main program with a set of coordinates which describe a closed figure. This set of points is then sent to the GROUND subroutine which calculates the area enclosed by the closed figure. This sequence of CLOSELOOP and GROUND is repeated until the end of the first line segment is found. The results are then returned to the user and the program is

ended.

The structure of the program can be described by the following set of Pascal-like statements.

```
begin
  get parameters from user and initialize variables
  repeat
    find set of coordinates for closed figure (CLOSELOOP)
    compute area enclosed by closed figure (GROUND)
    add area of closed figure to total area error
  until end of line segment is reached
  output results to user
end
```

The computation of the total length of the line drawing and the coded version of the line drawing is done within CLOSELOOP. This program is not very fast (i.e. do not expect an answer immediately).

The CLOSELOOP and GROUND subroutines can be broken down further. The GROUND subroutine implements the algorithm described in the preceding chapter almost exactly. The only difference is that a line parallel to the y-axis is used as the reference instead of the x-axis. This line has an x coordinate of the minimum x value in the closed figure. This is done to minimize the magnitude of the areas of the individual trapezoids. The minimization is done in order to maintain the highest level of accuracy possible on the microcomputer. The actual code is very straight forward and the reader is referred to the source listing for additional details. The implementation of CLOSELOOP was not as

straight forward and the discussion here will center on its implementation.

The CLOSELOOP subroutine can be described by the following Pascal-like statements.

```
begin
  repeat
    determine which line is ahead (digitized line
            drawing or coded version of line drawing)
    if digitized line drawing is ahead
     then get a point from the coded version file
     else get a point from the digitized line file
        (whenever a point is read from a file the
          length of that line is updated)
    determine if the two lines have intersected
  until the two lines intersect
end (return with a set of coordinates describing a
      closed figure)
```

These functions will now be broken down further.

The "determine which line is ahead" function is done by first determining the general direction (positive or negative along the x and y axis) that the coded version of the line is going in. If the digitized version of the line is farther along in the x or y directions than the coded version of the line then the digitized version is said to be ahead. This function is implemented as a function (DIGAHEAD) which returns a boolean value of true when the digitized version is ahead and a value of false otherwise.

The "get a point from the digitized line file" and "get a point from the coded version file" functions are identical except for which file they access. Immediately after a

point is read from either file the accumulated length of the appropriate line is updated. The final computed lengths will be output to the user and used in the area error per unit length calculation.

The "determine if the two lines intersect" function is not as simple as the two functions described previously. A Pascal-like description of this function is shown below.

```
begin
  initialize pointer into digitized point arrays
  repeat
    let segment1 = digitized(i) and digitized(i+1)
    increment i (digitized points pointer)
    initialize pointer into coded point array
    repeat
      let segment2 = coded(j) and coded(j+1)
      increment j (coded points pointer)
      compute possible intersection point of segment1
           and segment2
      determine if the intersection point is on both
           segments
    until an intersection is found or all coded segments
      are checked
  until an intersection is found or all digitized segments
    are checked
end
```

Most of the functions in the "determine if the two lines intersect" function just described are simple. The initialization is done such that the program does not check every combination of segments but checks only the last three segments of the coded line and the digitized line. This is

done only so the program will not run an excessively long time. The determination of whether or not a given point is on both line segments is done with a large if statement. The "compute possible intersection point of segment1 and segment2" is more complex and will be broken down further.

The "compute possible intersection point of segment1 and segment2" function is described below.

```
begin

  compute the delta x and delta y of segment1

  if delta x > delta y
   then use the points as given
   else rotate the figure 90 degrees
        (exchange the x and y axis values)

  if delta x = delta y
   then compute the delta x and delta y for segment2
        if delta x > delta y
         then use the points as given
         else rotate the figure 90 degrees

  if the delta x for either line segment is zero
   then compute the y value of the possible intersection
          point given one segment has infinite slope
   else compute the intersection point from the computed
          slopes and y-intercepts

end
```

The rotation of 90 degrees, when indicated, is done so that the computed slopes will practically always have a magnitude of less than one. This is done to maintain accuracy. If the machine had a high enough precision in the real numbers it uses then the rotation would not be necessary. The problem arises when the slope is very large and enough accuracy is lost due to the limitations of the machine that an intersection point that should have been found is not found. The section that computes the y value of a possible

intersection given that at least one segment has an infinite slope must also check to see if the delta x of both line segments are zero (can only occur if both line segments are really points and are both the same point). If this condition arises then we have the intersection point immediately, otherwise the y value is found using the computed slope and y intercept of the line that does not have an infinite slope. The computation of an intersection point given that neither slope is infinite is simple matter of geometry. The only case that must be checked before the intersection point is computed in this last case is where the slope of both line segments is the same. If the slopes are equal and the line segments overlap then the point farthest along in the direction the lines are going that is still on both line segments is chosen as the intersection point. If the slopes are equal and the line segments do not overlap then there is no intersection point and a flag is set to indicate the same. For any additional details on the implementation in program code of the error area computation the reader is referred to the source listings.

The area error computation program is designed to be interactive with the user. It prompts the user for all of the input it needs (the names of the two input files). The program is currently restricted in that there must be a completed closed figure before 58 points have been read from either file since the last closed figure. The only reason for this is the size of the arrays used to hold the data

points and the arrays could be easily enlarged as long as there is memory available.

One fact that should be pointed out again is that this program does not require the "coded version of the line drawing" to be any particular coded version of the line drawing. The only restriction is that the coded version of the line intersect the line drawing at least every 58 points. This property makes this area error computation program very general. Examples of the results obtained using this program can be found in the following chapter.

Summary

This chapter described the implementation of the chain coding algorithm and the area error computation. Both programs were explained from a functional point of view. The chain coding program was briefly explained and an example of the output of the chain coding program was given. The description of the area error computation program was more detailed and the description was done using a top down approach.

VI. Results and Recommendations

This chapter contains a demonstration of the usage of the set of programs that have been described in this thesis and contains the recommendations that I have for continued work with this system. First, a demonstration of the process of digitizing a line drawing, creating a set of chain coded versions of the line drawing, computing the area error per unit length, and then plotting the digitized line drawing and its chain coded versions will be given. Then I will make my recommendations for continued work with the system.

Using the System

This section contains a demonstration of the system. Since the system works mainly with graphical data this section contains a set of graphs showing the original line drawing, the digitized version of the line drawing, and coded versions of the digitized line drawing. The first of these graphs is shown in Figure 6-1.

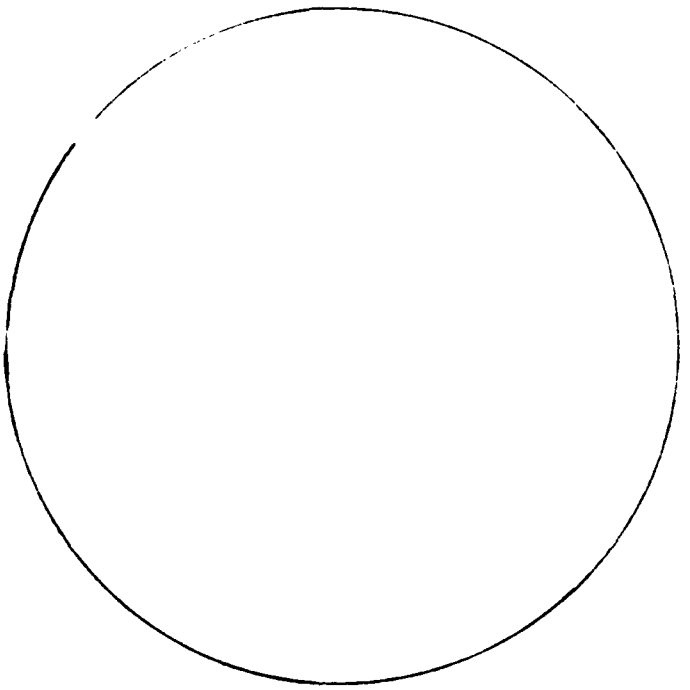


Figure 6-1 Original Line Drawing

This line drawing was placed on the digitizer and digitized using the DIGITIZE program. A plot of the digitized line drawing is shown in Figure 6-2. The plot was done using the PLOTFILE program with a scaling factor of 0.2 (scaled such that one inch on the digitizer equals one inch on the plotter).

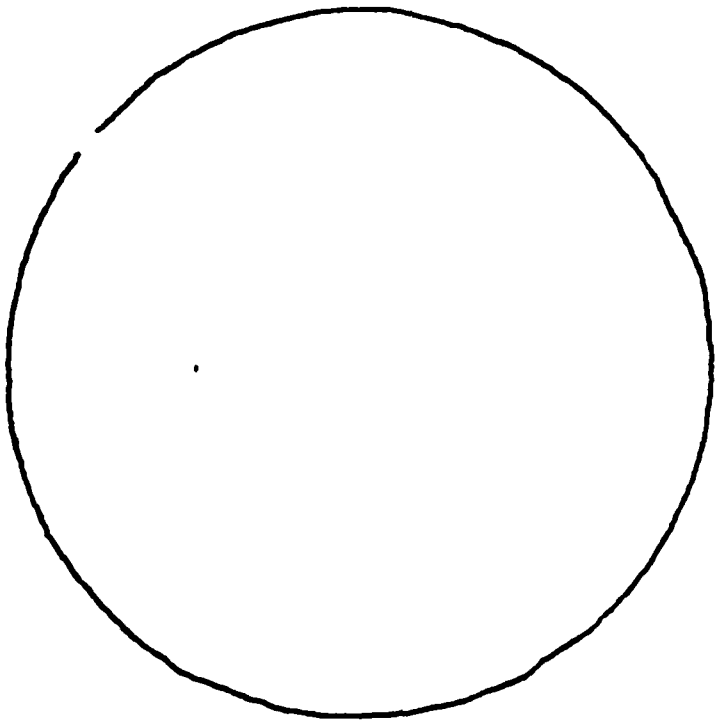
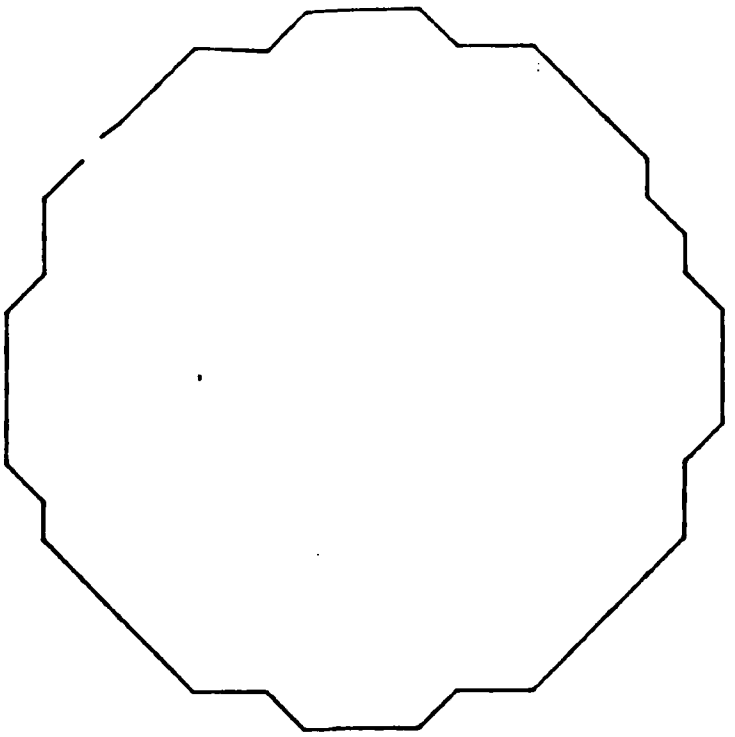


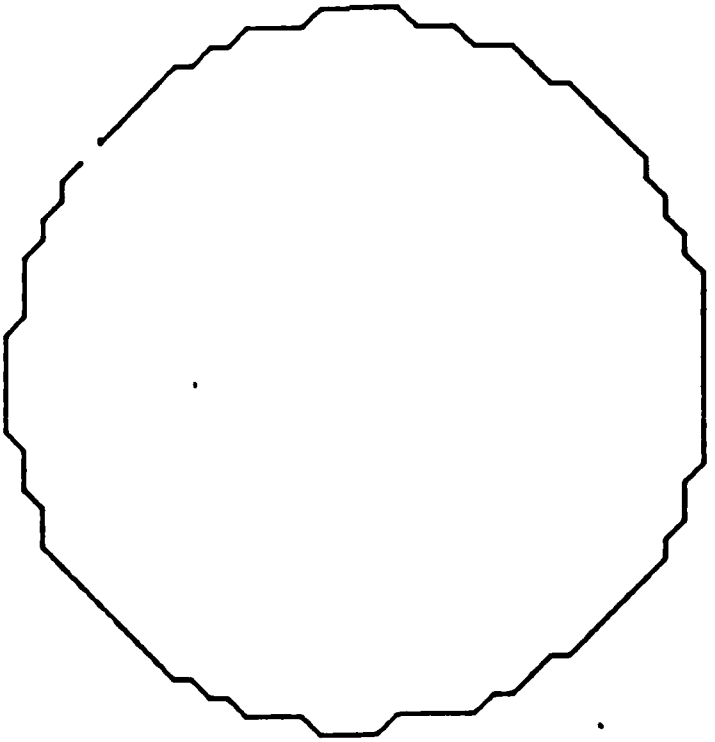
Figure 6-2 Digitized Version of Original Line Drawing

The digitized version of the line drawing was then input to the CODER program to generate chain coded versions of the line drawing. Four chain coded versions of the line drawing were done and the results are shown in Figures 6-3 through 6-6.



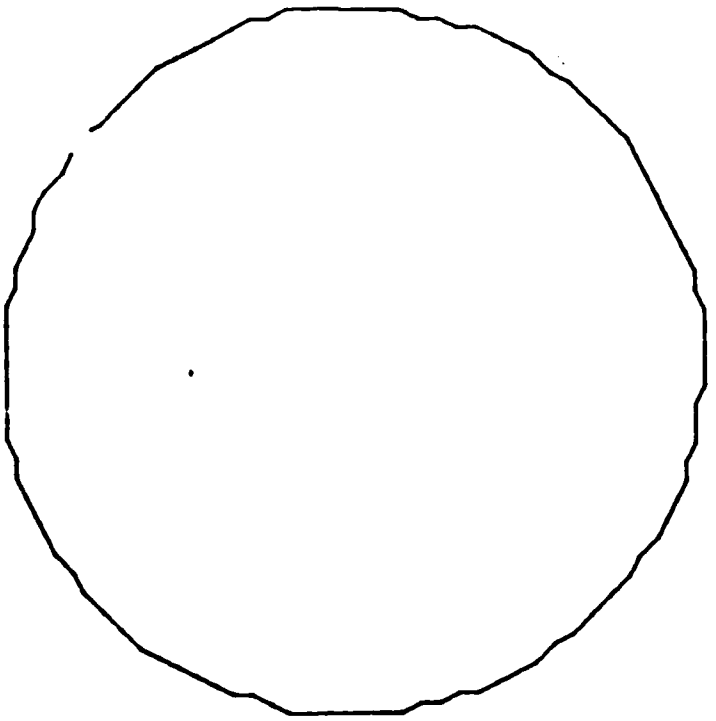
Gridsize = 0.20 inch Level = 1

Figure 6-3 Chain Coded Version 1 of Line Drawing



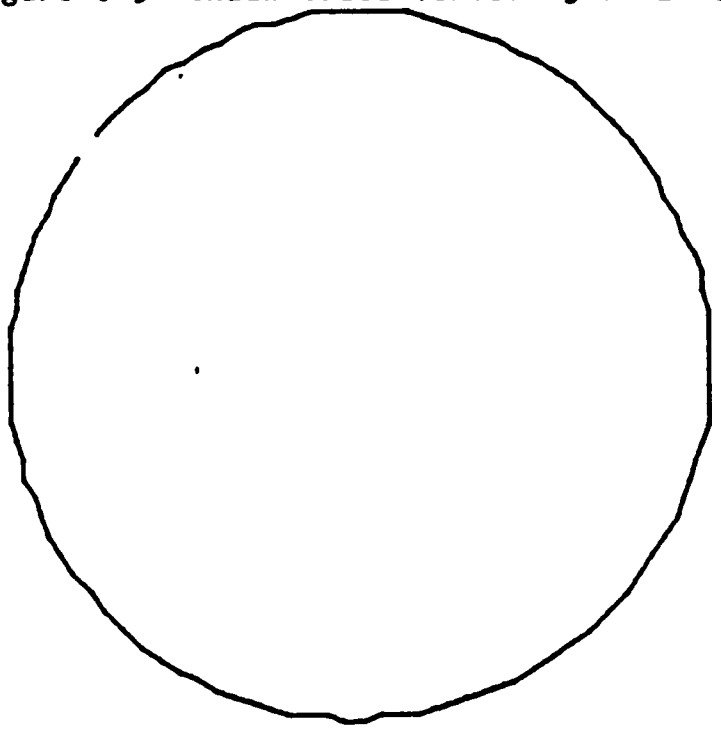
Gridsize = 0.10 inch Level = 1

Figure 6-4 Chain Coded Version 2 of Line Drawing



Gridsize = 0.05 inch Level = 2

Figure 6-5 Chain Coded Version 3 of Line Drawing



Gridsize = 0.033 inch Level = 3

Figure 6-6 Chain Coded Version 4 of Line Drawing

Figures 6-2 through 6-6 were done using a scaling factor of 0.2.

Now the ERROR program was run to obtain the performance measures shown in Figure 6-7.

Coded Version Level	Gridsize	Area Error/unit length
1	0.20	see note
2	0.10	0.6286 sq in/ 11.59 in
3	0.05	0.09693 sq in/ 11.59 in
4	0.033	0.06871 sq in/ 11.59 in

Figure 5-7 Performance of Chain Codes on Line Drawing

note: The area error could not be computed for this code because the array space allocated for points on the digitized line before an intersection was exceeded. This restriction can be overcome by devising a way to force an intersection. At the current time, the program only notifies the user if this error occurs and exits the program.

Recommendations

The following is a list of recommendations that I have for continued work on this system.

1.) Design a systematic method to evaluate the perfomance of different line drawings of given curvature characteristics to determine the impact of having the lines input from a digitizer. Inputting the line drawings in this manner introduces a random factor of where the line drawing is in relation to the origin and the digitized version of the line drawing will not have the same points that would be generated by a program simulating this input.

2.) Take the CODER and ERROR programs and make them subroutines to a main program. This main program would use a file of gridsize plus level specification as input and the output would be a file of the performance measures for the

chain codes specified. This would not be a difficult program to create but it would enable someone to plot the bits/unit length versus area error/unit length for a given line.

3.) Create a program that would compute the higher order chain codes. This program could be incorporated into the program described in 2.

4.) Obtain the upgrade kit from Heathkit to make the H89 microcomputer run at 4 MHz instead of 2 MHz. The reason for this recommendation will become very apparent if the program described in 2 is written. Currently such a program could easily run continuously all day before finishing.

BIBLIOGRAPHY

1. Freeman,Herbert. The Generalized Chain Code for Map DataEncodingand Processing. Technical ReportCRL-59. Air Force Office of Scientific Research. June 1978.

2. Freeman, Herbert. "Computer Processing of Line Drawing Images", Computer Surveys. Volume 6, number 1, pp 57-59. March 1974.

3. Hewlett-Packard 9874A Documentation. Hewlett-Packard Desktop Computer Division, Fort Collins, CO. August 1978.

4. HIPLOT DIGITAL PLOTTERS OPERATOR'S INSTRUCTIONS. Bausch & Lomb Houston Instrument Division, Austin TX. 1981.

5. Jones, Keith R. "Grid-Based Line Drawing Quantization." Master's Thesis AFIT/GE/EE/82D-41. Air Force Institute of Technology, Wright Patterson Air Force Base, Ohio. Dec 1982.

6. Model 4885A RS-232 to IEEE 488 Controller Instruction Manual. ICS Electronics Corporation, San Jose, CA. January 1979.

7. Moffitt, P. C. and Bouchard, H. Surveying, 6th edition. Intext Educational Publishers, New York NY, 1975. pp 307-320.

8. Sagrhi, John A. "Efficient Encoding of Line Drawing Data with Generalized Chain Codes." Rensselaer Polytechnic Institute, Troy, Ney York. August 1979.

Appendix A

Source Listing of Interfacing Routines

This appendix contains the source listings for the low level, hardware dependant routines. The listings are documented with comments throughout.

These routines were in the file BUSLIB.MAC.

```
          .Z80
;   BUS I/O ROUTINES  (TO TALK TO IEEE 488 BUS CONTROLLER)
;    WRITTEN BY JOSEPH E. ROCK
;      DATE : 9 MARCH 1983
;
;   EQUATES FOR EASE OF UNDERSTANDING CODE
BOOT      EQU     0000H       ; CP/M BOOT ADDRESS
DATA      EQU     320Q        ; SERIAL PORT DATA REG ADDR
CONPORT   EQU     350Q        ; CONSOLE I/O PORT ADDRESS
RDA       EQU     01          ; RECEIVED DATA AVAILABLE
TBE       EQU     20H         ; TRANSMITTER BUFFER EMPTY
CR        EQU     015Q        ; CARRAGE RETURN
LF        EQU     10          ; LINE FEED CHAR
CNTRLC    EQU     03          ; CONTROL C
;
;
;
          PUBLIC BUSIN,BUSOUT,BUSINT
;
;     BUSINT INITIALIZES THE SERIAL PORT TO ESTABLISH
;COMMUNICATIONS WITH THE IEEE 488 BUS CONTROLLER AND THEN
; INITIALIZES THE CONTROLLER AND THE DIGITIZER. CALLED FROM
; THE MAIN PROG AS  'BUSINT:CHAR' AND RETURNS A VALUE OF 'E'
; IF ANY ERRORS OCCUR (OTHERWISE IT RETURNS  A '0').
;
BUSINT:   POP     DE          ; SAVE RETURN ADDRESS
;
;   INITIALIZE UART AT 'DATA' PORT
;
          LD      A,0         ; CLEAR INTRPT ENABLE REG
          OUT     (DATA+1),A
          LD      A,10H       ; SET FOR LOOP OPERATION
          OUT     (DATA+4),A
          LD      A,80H       ; ENABLE BAUD SET
          OUT     (DATA+3),A
          LD      A,014Q      ; PICK 9600 BAUD RATE
          OUT     (DATA),A
          LD      A,0
          OUT     (DATA+1),A
          LD      A,03H       ; 8BIT,1STOP,NO PARITY
          OUT     (DATA+3),A
          IN      A,(DATA)    ; READ ANY GARBAGE AND DISCARD
;
          LD      A,20        ; DELAY 2 300 BAUD CHARS
BINT1:    PUSH    AF          ; 70 MSEC
```

```
          LD      A,255
BINT2:    DEC     A             ; 3.5 MSEC
          JP      NZ,BINT2
          POP     AF
          DEC     A
          JP      NZ,BINT1
;
          IN      A,(DATA)      ; READ ANY GARBAGE AND DISCARD
          LD      A,0           ; TAKE OUT OF LOOP OPERATION
          OUT     (DATA+4),A
;
;  INTIALIZE BUS CONTROLLER.. SEND IT '@@@'
          LD      HL,LINE1      ; CONTAINS '@@@' AND
                                ;   END OF LINE MARK ('CR')
          CALL    LINOT         ; OUTPUTS LINE TO CONTROLLER
;  NOW SEE IF CONTROLLER HAS ANYTHING TO SAY
          LD      A,20          ; WE'LL WATCH FOR A WHILE
BINT3:    PUSH    AF
          LD      A,255
BINT4:    PUSH    AF
          IN      A,(DATA+5)    ; ? CHAR AVAILABLE
          AND     RDA           ;        FROM CONTROLLER
          JP      NZ,BINTE      ; IF WE DO THEN WE HAVE AN ERROR
          IN      A,(CONPORT+5) ; ? CHAR AVAILABLE
          AND     RDA           ;    FROM CONSOLE
          JP      NZ,BINT5      ; NO.. JUMP AROUND
          IN      A,(CONPORT)   ; YES .. GET CHAR
          AND     07FH          ; MASK OUT MSB
          AND     CNTRLC        ; ? CONTROL C
          JP      Z,BINTE       ; IF YES THEN END WITH
                                ;      ERROR INDICATION
BINT5:    POP     AF
          DEC     A
          JP      NZ,BINT4
          POP     AF            ; GET FIRST COUNTER BACK
          DEC     A
          JP      NZ,BINT3
;  INITIALIZE DIGITIZER.. SEND THE DIGITIZER 'IN'
          LD      HL,LINE2
          CALL    LINOT         ; SENDS '@WRT 06,IN (CR)'
                                ;     TO CONTROLLER
; PREPARE TO RETURN
          LD      C,'0'         ; RETURN '0' FOR NORMAL EXIT
          JP      BINTD
BINTE:    LD      C,'E'         ; RETURN 'E' FOR ERROR INDICATION
; READY TO GO BACK
BINTD:    LD      B,0           ; HIGH BYTE OF CHAR ON STACK IS 00
          PUSH    BC            ; PUT VALUE ON STACK
          PUSH    DE            ; PUT RETURN ADDRESS BACK ON STACK
          RET                   ; ALL DONE
; OUTPUT LINES :
LINE1:    DB      '@@@',CR,LF
LINE2:    DB      '@WRT 06,IN',CR,LF
;
;
```

```
;       BUSIN IS USED TO INPUT A LINE FROM A DEVICE ON THE
;  IEEE 488 BUS VIA THE BUS CONTROLLER.  TO CALL THE ROUTINE
;  USE
;  'BUSIN( DEVICE:INTEGER; VAR ERFLAG:CHAR;
;                   VAR LINE:ARRAY[1,40] OF CHAR);'
;  WHERE DEVICE IS TWO CHARACTERS IN THE INTEGER VARIABLE
;     THAT DENOTE THE DEVICE NUMBER ('06' FOR THE DIGITIZER)
;        ERFLAG IS ONE CHAR THAT RETURNS AS A 'E'
;          IF AN ERROR IS ENCOUNTERED
;            OR A "CONTROL C" IS ENTERED FROM THE KB
;        IF ERFLAG = 'E' UPON ENTRY IT WILL SIMPLY
;                                RETURN TO THE USER
;        IF NO ERRORS OCCUR THEN ERFLAG WILL RETURN AS '0'
;        LINE IS THE ARRAY THAT THE CHARS FROM THE BUS
;         CONTROLLER WILL BE PLACED (LAST CHAR WILL BE A 'CR')
;
BUSIN:   POP     DE          ; GET RETURN ADDRESS
         POP     HL          ; GET ADDRESS OF ARRAY
         LD      (ARRAY),HL  ; SAVE ARRAY ADDRESS
         POP     HL          ; GET ERFLAG ADDRESS
         POP     BC          ; GET DEVICE NUMBER
         PUSH    DE          ; PUT RETURN ADDRESS BACK ON STACK
; CHECK ICF ERFLAG = 'E'
         LD      A,(HL)      ; GET ERFLAG VALUE
         CP      'E'         ; ? =
         JP      Z,BIER      ; YES, JUMP TO BUSIN ERROR SECTION
         LD      (ERFLAG),HL ; SAVE ERFLAG ADDRESS
; NO ERROR, GET DATA FROM BUS CONTROLLER
; FIRST COMPLETE THE LINE TO BE OUTPUT TO CONTROLLER
         LD      A,B         ; FIRST DIGIT
         LD      (BILN2),A   ; PUT FIRST DIGIT IN LINE
         LD      A,C         ; SECOND DIGIT
         LD      (BILN2+1),A  ; PUT SECOND DIGIT IN LINE
; TURN OFF KB INTERRUPT (MONITOR THE KB HERE)
         LD      A,0
         OUT     (CONPORT+1),A
; RESET ERFLAG VALUE FOR NORMAL EXIT
         LD      BC,(ERFLAG) ; GET ERFLAG ADDRESS BACK
         LD      A,'0'       ; ERFLAG VALUE IF NO ERROR
         LD      (BC),A      ; PUT '0' IN CALLER'S ERFLAG
; TELL BUS CONTROLLER TO READ FROM DEVICE AND SEND TO YOU
         LD      HL,BILN1    ; '@RED (DEVICE NUMBER)(CR)'
         CALL    LINOT       ; OUTPUT LINE
; MONITOR KB FOR (CONTROL C) AND BUS CONTROLLER FOR ANY INPUT
         LD      HL,(ARRAY)  ; GET ARRAY ADDRESS INTO A
                             ;         USABLE PLACE
; BEGIN KB AND BUS CONTROLLER MONITORING
BILP1:   IN      A,(CONPORT+5)  ; ? CHAR AVAILABLE
         AND     RDA         ;              FROM CONSOLE
         JP      Z,BILP2     ; NO CHAR
         IN      A,(CONPORT) ; GET CHAR FROM CONSOLE
         AND     07FH        ; MASK OUT MSB
         CP      CNTRLC      ; ? CONTROL C
         JP      Z,BIER      ; YES, SET ERFLAG TO 'E' AND RETURN
BILP2:   IN      A,(DATA+5)  ; ? CHAR AVAILABLE
```

```
          AND     RDA          ;         FROM BUS CONTROLLER
          JP      Z,BILP1      ; NO CHAR, TRY AGAIN
          CALL    RDBUS        ; YES, PUT CHAR IN ARRAY
                               ;     (RETURN WITH CHAR IN A)
          CP      CR           ; IF CR THEN WE HAVE ENTIRE LINE
          JP      NZ,BILP1     ; NOT A CR, TRY AGAIN
; WE NOW HAVE LINE SO PREPARE TO EXIT
          LD      HL,(ARRAY)   ; GET ARRAY ADDRESS BACK
          LD      A,(HL)       ; GET FIRST CHAR OF ARRAY
          CP      'E'          ; IS DATA AN ERROR MESSAGE
          JP      NZ,BINX1     ; NO, LEAVE ERFLAG ALONE
BIER:     LD      HL,(ERFLAG)  ; GET ERFLAG ADDRESS BACK
          LD      (HL),'E'     ; ALERT MAIN PROG TO ERROR
; REACTIVATE KB INTERRUPT AND EXIT
BINX1:    LD      A,01         ; CONSOLE INTERRUPT RESTORE
          OUT     (CONPORT+1),A ; RESTORE KB INTERRUPT
          RET                  ; ALL DONE
; OUTPUT LINES :
ARRAY:    DS      2            ; PLACE TO KEEP ARRAY ADDRESS
ERFLAG:   DS      2            ; PLACE TO KEEP ERFLAG ADDRESS
BILN1:    DB      '@RED '      ; FIRST PART OF READ COMMAND
BILN2:    DB      '06',CR      ; LAST HALF OF READ COMMAND
                               ;       (DEVICE NO = 06)
;
;
;       BUSOUT IS USED TO WRITE A LINE TO A DEVICE ON THE
;   IEEE 488 BUS VIA THE BUS CONTROLLER.  TO CALL THE ROUTINE
;   USE:
;      'BUSOUT( DEVICE:INTEGER; VAR ERFLAG:CHAR;
;                          VAR LINE:ARRAY[1,40] OF CHAR);'
;    WHERE:  DEVICE IS TWO ASCII CHARS IN THE INTEGER THAT
;               DENOTE THE DEVICE NUMBER ('06' FOR DIGITIZER)
;            ERFLAG IS ONE CHAR THAT RETURNS AS 'E'
;               IF AN ERROR IS ENCOUNTERED DURING EXECUTION
;             - IF ERFLAG = 'E' UPON ENTRY THEN BUSOUT
;                    WILL SIMPLY RETURN
;             - IF NO ERRORS ARE ENCOUNTERED THEN ERFLAG = '0'
;            UPON RETURN LINE IS THE ARRAY OF CHAR TO BE SENT
;            TO THE SPECIFIED DEVICE (LAST CHAR OF LINE
;                    MUST BE A 'CR')
;
BUSOUT:   POP     DE           ; GET RETURN ADDRESS OFF STACK
          POP     HL           ; GET ADDRESS OF ARRAY
          LD      (ARRAY),HL   ; SAVE ARRAY ADDRESS
          POP     HL           ; GET ERFLAG ADDRESS
          POP     BC           ; GET DEVICE NUMBER
          PUSH    DE           ; PUT RETURN ADDRESS BACK ON STACK
; CHECK IF ERFLAG = 'E'
          LD      A,(HL)       ; GET ERFLAG VALUE
          CP      'E'          ; ? ERROR UPON ENTRY
          JP      Z,BOTER      ; GO TO ERROR SECTION
; NO ERROR, OUTPUT LINE
          LD      (ERFLAG),HL  ; SAVE ERFLAG ADDRESS
          LD      HL,LIOUT2    ; PREPARE TO PUT DEVICE
                               ;       NO. IN OUTPUT LINE
```

```
          LD      (HL),B      ; PUT FIRST DIGIT IN LINE
          INC     HL          ; POINT TO NEXT LOCATION
          LD      (HL),C      ; PUT SECOND DIGIT IN LINE
; OUTPUT FIRST HALF OF LINE
          LD      HL,LIOUT1   ; POINT TO FIRST PART OF LINE
          CALL    LINOUT      ; OUTPUT CHARS
; OUTPUT SECOND HALF OF LINE (DEVICE COMMAND)
          LD      HL,(ARRAY)  ; POINT TO REST OF LINE
          CALL    LINOT       ; OUTPUT LINE
; COMMAND HAS BEEN SENT
; PREPARE TO EXIT
          LD      HL,(ERFLAG) ; GET ERFLAG ADDRESS BACK
          LD      (HL),'0'    ; NORMAL EXIT
BOTER:    RET                 ; ALL DONE
;
; SPECIAL OUTPUT ROUTINE (QUITS ON '}' CHAR)
LINOUT:   LD      A,(HL)      ; GET CHAR
          CP      '}'         ; END OF OUTPUT
          RET     Z           ; RETURN IF SO
          CALL    BUSOT       ; OUTPUT CHAR
          INC     HL          ; POINT TO NEXT CHAR
          JP      LINOUT      ; AND CONTINUE
; OUTPUT LINES
LIOUT1:   DB      '@WRT '     ; FIRST HALF OF BUS COMMAND
LIOUT2:   DB      '06,}'      ; LAST HALF OF BUS COMMAND
;
;
; OUTPUT LINE TO CONTROLLER ROUTINE
;
LINOT:    LD      A,(HL)      ; GET CHAR FROM BUFFER
          CALL    BUSOT       ; OUTPUT CHAR TO CONTROLLER
          CP      CR          ; END OF LINE ?
          RET     Z           ; YES, STOP SENDING CHARS
          INC     HL          ; POINT TO NEXT CHAR
          JP      LINOT       ; OUTPUT NEXT CHAR
;
;  READ CHAR FROM BUS CONTROLLER ROUTINE
;
RDBUS:    IN      A,(DATA)    ; GET INPUT CHAR FROM CONTROLLER
          AND     07FH        ; MASK OUT MSB
          LD      (HL),A      ; STORE INPUT CHAR IN BUFFER
          INC     HL          ; INCREMENT BUFFER POINTER
          RET                 ; ALL DONE
;
;  OUTPUT CHAR TO IEEE BUS CONTROLLER
;     OUTPUTS CHAR IN REG A
;     NOTHING ELSE AFFECTED
BUSOT:    PUSH    AF          ; SAVE OUTPUT CHAR
BSLOOP:   IN      A,(DATA+5)  ; GET STATUS OF PORT
          AND     TBE         ; TRANSMIT BUFFER EMPTY ???
          JP      Z,BSLOOP    ; NO, TRY AGAIN
          POP     AF          ; GET CHAR BACK
          OUT     (DATA),A    ; OUTPUT CHAR
          RET                 ; ALL DONE
          END
```

The following routines were in the file PLTLIB.MAC.

```
        .Z80
;  SERIAL PORT ROUTINES  (TO TRANSMIT AND RECEIVE DATA
;                              FROM PORT 330-337)
;   WRITTEN BY JOSEPH E. ROCK
;     DATE STARTED: 17 JUNE 1983
;
;  EQUATES FOR EASE OF UNDERSTANDING CODE
BOOT    EQU     0000H       ; CP/M BOOT ADDRESS
DATA    EQU     330Q        ; SERIAL PORT DATA REG ADDR
CONPORT EQU     350Q        ; CONSOLE I/O PORT ADDRESS
RDA     EQU     01          ; RECEIVED DATA AVAILABLE
TBE     EQU     20H         ; TRANSMITTER BUFFER EMPTY
CR      EQU     015Q        ; CARRAGE RETURN
LF      EQU     10          ; LINE FEED CHAR
CNTRLC  EQU     03          ; CONTROL C
;
;
;
        PUBLIC PORTIN,CHARIN,CHAROT,LINOUT
;
;    PORTIN IS USED TO INITIALIZE THE SERIAL PORT AT THE
;  'PORT' ADDRESS THE ROUTINE IS ONLY CALLED ONCE DURING
;  A PROGRAM. TO CALL THE ROUTINE USE:
;    'PORTIN'  -- THE ROUTINE HAS NO PARAMETERS.
;
;
PORTIN: POP     DE          ; SAVE RETURN ADDRESS
;
;  INITIALIZE UART AT 'DATA' PORT
;
        LD      A,0         ; CLEAR INTRPT ENABLE REG
        OUT     (DATA+1),A
        LD      A,10H       ; SET FOR LOOP OPERATION
        OUT     (DATA+4),A
        LD      A,80H       ; ENABLE BAUD SET
        OUT     (DATA+3),A
        LD      A,014Q      ; PICK 9600 BAUD RATE
        OUT     (DATA),A
        LD      A,0
        OUT     (DATA+1),A
        LD      A,03H       ; 8BIT,1STOP,NO PARITY
        OUT     (DATA+3),A
        IN      A,(DATA)    ; READ ANY GARBAGE AND DISCARD
;
        LD      A,20        ; DELAY 2 300 BAUD CHARS
```

```
PORT1:    PUSH    AF          ; 70 MSEC
          LD      A,255
PORT2:    DEC     A           ; 3.5 MSEC
          JP      NZ,PORT2
          POP     AF
          DEC     A
          JP      NZ,PORT1
;
          IN      A,(DATA)    ; READ ANY GARBAGE AND DISCARD
          LD      A,0         ; TAKE OUT OF LOOP OPERATION
          OUT     (DATA+4),A
;
; READY TO GO BACK
          PUSH    DE          ; PUT RETURN ADDRESS BACK ON STACK
          RET                 ; ALL DONE
;
;
;       CHARIN IS USED TO INPUT A CHARACTER FROM THE SERIAL
;  PORT. TO CALL THE ROUTINE USE:
;   'CHARIN(VAR INPUT:CHAR; VAR ERFLAG:CHAR);
;         INPUT IS THE CHARACTER RECEIVED FROM THE PORT
;         IF ERFLAG = 'E' UPON ENTRY IT WILL SIMPLY RETURN
;                                TO THE USER
;         IF NO ERRORS OCCUR THEN ERFLAG WILL RETURN AS '0'
;           ELSE ERFLAG = 'E' UPON EXIT
;
CHARIN:   POP     DE          ; GET RETURN ADDRESS
          POP     HL          ; GET ADDRESS OF ERFLAG
          POP     BC          ; GET ADDRESS OF INPUT VARIABLE
          LD      (CHIN),BC   ; SAVE INPUT ADDRESS
          PUSH    DE          ; PUT RETURN ADDRESS BACK ON STACK
; CHECK IF ERFLAG = 'E'
          LD      A,(HL)      ; GET ERFLAG VALUE
          CP      'E'         ; ? =
          JP      Z,CIER      ; YES, JUMP TO CHARIN ERROR SECTION
          LD      (ERFLAG),HL ; SAVE ERFLAG ADDRESS
; NO ERROR, GET DATA FROM PORT
; TURN OFF KB INTERRUPT (MONITOR THE KB HERE)
          LD      A,0
          OUT     (CONPORT+1),A
; RESET ERFLAG VALUE FOR NORMAL EXIT
          LD      BC,(ERFLAG) ; GET ERFLAG ADDRESS BACK
          LD      A,'0'       ; ERFLAG VALUE IF NO ERROR
          LD      (BC),A      ; PUT '0' IN CALLER'S ERFLAG
; MONITOR KB FOR (CONTROL C) AND PORT FOR ANY INPUT
          LD      HL,(CHIN)   ; GET ARRAY ADDRESS INTO A
                              ;         USABLE PLACE
; BEGIN KB AND BUS CONTROLLER MONITORING
CILP1:    IN      A,(CONPORT+5)  ; ? CHAR AVAILABLE
          AND     RDA         ;              FROM CONSOLE
          JP      Z,CILP2     ; NO CHAR
          IN      A,(CONPORT) ; GET CHAR FROM CONSOLE
          AND     07FH        ; MASK OUT MSB
          CP      CNTRLC      ; ? CONTROL C
          JP      Z,CIER      ; YES, SET ERFLAG TO 'E' AND RETURN
```

```
CILP2:    IN      A,(DATA+5) ; ? CHAR AVAILABLE
          AND     RDA        ;         FROM BUS CONTROLLER
          JP      Z,CILP1    ; NO CHAR, TRY AGAIN
          IN      A,(DATA)   ; YES, GET CHAR FROM PORT
          AND     07FH       ; MASK OUT MSB
          LD      (HL),A     ; STORE INPUT CHAR IN BUFFER
; WE NOW HAVE CHAR SO PREPARE TO EXIT
          JP      CINX1      ; LEAVE ERFLAG ALONE
CIER:     LD      HL,(ERFLAG) ; GET ERFLAG ADDRESS BACK
          LD      (HL),'E'   ; ALERT MAIN PROG TO ERROR
; REACTIVATE KB INTERRUPT AND EXIT
CINX1:    LD      A,01       ; CONSOLE INTERRUPT RESTORE
          OUT     (CONPORT+1),A ; RESTORE KB INTERRUPT
          RET                ; ALL DONE
; STORAGE AREA :
CHIN:     DS      2          ; PLACE TO KEEP ARRAY ADDRESS
ERFLAG:   DS      2          ; PLACE TO KEEP ERFLAG ADDRESS
;
;
;       CHAROUT IS USED TO OUTPUT A SINGLE CHARACTER FROM
;   THE PORT. TO CALL THE ROUTINE USE :
;      'CHAROUT(VAR OUTPUT:CHAR; VAR ERFLAG:CHAR; );'
;    WHERE:  OUTPUT IS THE ADDRESS OF THE CHARACTER TO BE
;              OUTPUT TO THE PORT
;            ERFLAG IS ONE CHAR THAT RETURNS AS 'E'
;               IF AN ERROR IS ENCOUNTERED DURING EXECUTION
;             - IF ERFLAG = 'E' UPON ENTRY THEN BUSOUT
;                                 WILL SIMPLY RETURN
;             - IF NO ERRORS ARE ENCOUNTERED THEN
;                               ERFLAG = '0' UPON RETURN
;
CHAROT:   POP     DE         ; GET RETURN ADDRESS OFF STACK
          POP     HL         ; GET ERFLAG ADDRESS
          LD      (ERFLAG),HL ; SAVE ERFLAG ADDRESS
          POP     BC         ; GET ADDRESS OF OUTPUT
          LD      (CHOUT),BC ; SAVE ARRAY ADDRESS
          PUSH    DE         ; PUT RETURN ADDRESS BACK ON STACK
; CHECK IF ERFLAG = 'E'
          LD      A,(HL)     ; GET ERFLAG VALUE
          CP      'E'        ; ? ERROR UPON ENTRY
          JP      Z,COTER    ; GO TO ERROR SECTION
; NO ERROR, OUTPUT CHARACTER
          LD      HL,(CHOUT) ; POINT TO CHARACTER
          LD      A,(HL)     ; GET CHARACTER
          CALL    BUSOT      ; OUTPUT CHARACTER
; CHARACTER HAS BEEN SENT
; PREPARE TO EXIT
          LD      HL,(ERFLAG) ; GET ERFLAG ADDRESS BACK
          LD      (HL),'0'   ; NORMAL EXIT
COTER:    RET                ; ALL DONE
; STORAGE
CHOUT:    DS      2
;
;
;
```

```
;       LINOUT IS USED TO OUTPUT AN ARRAY OF CHARACTERS TO
;    THE PORT. THE ARRAY MUST TERMINATE WITH A } CHAR.
;    THE } CHAR IS NOT TRANSMITTED. TO CALL THE ROUTINE USE :
;    'LINOUT(VAR LINE:ARRAY[1..40] OF CHAR; VAR ERFLAG:CHAR; );'
;    WHERE:  LINE IS THE ADDRESS OF THE ARRAY OF CHARACTERS
;         TO BE OUTPUT TO THE PORT
;            ERFLAG IS ONE CHAR THAT RETURNS AS 'E' IF AN
;               ERROR IS ENCOUNTERED DURING EXECUTION
;             - IF ERFLAG = 'E' UPON ENTRY THEN BUSOUT
;                    WILL SIMPLY RETURN
;             - IF NO ERRORS ARE ENCOUNTERED THEN
;                          ERFLAG = '0' UPON RETURN
;
LINOUT:  POP     DE          ; GET RETURN ADDRESS OFF STACK
         POP     HL          ; GET ERFLAG ADDRESS
         LD      (ERFLAG),HL ; SAVE ERFLAG ADDRESS
         POP     BC          ; GET ADDRESS OF OUTPUT
         LD      (LIOUT),BC  ; SAVE ARRAY ADDRESS
         PUSH    DE          ; PUT RETURN ADDRESS BACK ON STACK
; CHECK IF ERFLAG = 'E'
         LD      A,(HL)      ; GET ERFLAG VALUE
         CP      'E'         ; ? ERROR UPON ENTRY
         JP      Z,LOTER     ; GO TO ERROR SECTION
; NO ERROR, OUTPUT LINE
         LD      HL,(LIOUT)  ; POINT TO CHARACTER
LILOOP:  LD      A,(HL)      ; GET CHAR
         CP      '}'         ; END OF OUTPUT
         JP      Z,LICONT    ; RETURN IF SO
         CALL    BUSOT       ; OUTPUT CHAR
         INC     HL          ; POINT TO NEXT CHAR
         JP      LILOOP      ; AND CONTINUE
; LINE HAS BEEN SENT
; PREPARE TO EXIT
LICONT:  LD      HL,(ERFLAG) ; GET ERFLAG ADDRESS BACK
         LD      (HL),'0'    ; NORMAL EXIT
LOTER:   RET                 ; ALL DONE
; STORAGE AREA :
LIOUT:   DS      2
;
;  OUTPUT CHAR TO SERIAL PORT ROUTINE
;     OUTPUTS CHAR IN REG A
;     NOTHING ELSE AFFECTED
BUSOT:   PUSH    AF          ; SAVE OUTPUT CHAR
BSLOOP:  IN      A,(DATA+5)  ; GET STATUS OF PORT
         AND     TBE         ; TRANSMIT BUFFER EMPTY ???
         JP      Z,BSLOOP    ; NO, TRY AGAIN
         POP     AF          ; GET CHAR BACK
         OUT     (DATA),A    ; OUTPUT CHAR
         RET                 ; ALL DONE
;
;
         END
```

This listing was contained in the file DIGRTNS.SRC.

```
MODULE DIGRTNS;
(* WRITTEN BY:  JOSEPH E. ROCK, JR.                                    *)
(*  THESE ROUTINES PROVIDE A RELATIVELY HIGH LEVEL METHOD TO           *)
(* INTERFACE TO THE DIGITIZER.                                         *)
(*  TO CALL THE GET_POINT ROUTINE USE:                                 *)
(*    GET_POINT(DATA_LINE,ERFLAG,KEYNUMBER);                           *)
(*     WHERE:                                                          *)
(*          DATA_LINE IS AN ARRAY[1..20] OF CHAR                       *)
(*          ERFLAG IS BOOLEAN                                          *)
(*          KEYNUMBER IS AN INTEGER                                    *)
(*    FUNCTIONAL DESCRIPTION:                                          *)
(*       1. IF KEYNUMBER < 0 THEN INITIALIZE SAMPLING PARAMETERS*)
(*       2. IF ERFLAG = 'E' THEN DO NOTHING MORE                       *)
(*       3. KEEP CHECKING DIGITIZER UNTIL SOMETHING HAPPENS            *)
(*       4. IF SPECIAL FUNCTION KEY ENTERED THEN                       *)
(*           IF Fa THEN SET SAMPLING PARAMETERS AND GOTO 3             *)
(*             ELSE SET KEYNUMBER AND RETURN                           *)
(*       5. IF ERROR CONDITION THEN GET ERROR INDICATOR                *)
(*           IF OUT-OF-BOUNDS THEN HANDLE IT AND GOTO 3                *)
(*             ELSE SEND MESSAGE, ERFLAG = 'E', AND RETURN             *)

TYPE DIGLINE = ARRAY[1..20] OF CHAR;

CONST
  DIGITIZER = 12342 ; (* 12342 = '06' - DIGITIZER DEVICE NUMBER *)

VAR
  DMODE,OUTKEY,CLRKEY,STATUS,OUTPNT,OUTERR : DIGLINE;
  RESPONSE : DIGLINE;

EXTERNAL PROCEDURE BUSIN(DEVICE:INTEGER; VAR ERFLAG:CHAR;
            VAR INLINE:DIGLINE);
EXTERNAL PROCEDURE BUSOUT(DEVICE:INTEGER; VAR ERFLAG:CHAR;
            VAR OUTLINE:DIGLINE);
EXTERNAL FUNCTION BUSINT:CHAR;
EXTERNAL PROCEDURE WHATNUM(VAR CHARAC:DIGLINE;
            VAR LENGTH,NUMB:INTEGER);

PROCEDURE GET_POINT(VAR DATA_LINE:DIGLINE; VAR ERFLAG:CHAR;
            VAR KEYNUMBER:INTEGER);

VAR
  RESPONSE : DIGLINE;
  NUMBER,LENGTH : INTEGER;
```

```
  FINISH : BOOLEAN;

FUNCTION PREPARE:CHAR;

VAR  ERFLAG : CHAR;

BEGIN
  ERFLAG := BUSINT;
  IF ERFLAG = 'E' THEN
    WRITELN('DIGITIZER INITIALIZATION FAILED... HELP...');
  STATUS[1] := 'O';
  STATUS[2] := 'S';
  STATUS[3] := CHR(13);
  OUTPNT[1] := 'O';
  OUTPNT[2] := 'D';
  OUTPNT[3] := CHR(13);
  OUTKEY[1] := 'O';
  OUTKEY[2] := 'K';
  OUTKEY[3] := CHR(13);
  CLRKEY[1] := 'S';
  CLRKEY[2] := 'K';
  CLRKEY[3] := ' ';
  CLRKEY[4] := 'O';
  CLRKEY[5] := ' ';
  CLRKEY[6] := CHR(13);
  OUTERR[1] := 'O';
  OUTERR[2] := 'E';
  OUTERR[3] := CHR(13);
  PREPARE := ERFLAG;
END;

PROCEDURE SAMPLING(VAR ERFLAG:CHAR);

VAR COUNTER,SAVECNT : INTEGER;

BEGIN
  WRITELN;
  WRITE('SINGLE OR CONTINUOUS SAMPLE MODE (SG/CN): ');
  READLN(DMODE[1],DMODE[2]);
  IF DMODE[1] = 'S' THEN DMODE[3] := CHR(13)
    ELSE
      BEGIN
        WRITELN('CONTINUOUS SAMPLING MODE SELECTED:');
        DMODE[3] := ' ';
        WRITELN('DEFAULTS ARE :');
        WRITELN('         DELTA d = 400 X 1E-3 in ');
        WRITELN('         DELTA t = 20 msec ');
        WRITE('  DELTA t = ?   (20 to 32767 msec) ',
                    '(CR FOR DEFAULT) :');
        COUNTER := 4;
        REPEAT
          READ(DMODE[COUNTER]);
          COUNTER := COUNTER + 1;
        UNTIL EOLN;
        IF DMODE[4] = ' ' THEN
```

```
            BEGIN
              DMODE[4] := '2';
              DMODE[5] := '0';
              COUNTER := 6;
            END;
          DMODE[COUNTER] := ',';
          COUNTER := COUNTER + 1;
          SAVECNT := COUNTER;
          WRITE(' DELTA d = ?  (1 TO 5000 X1E-3in.. 1=.001in)',
                    '(CR FOR DEFAULT) :');
          REPEAT
            READ(DMODE[COUNTER]);
            COUNTER := COUNTER + 1;
          UNTIL EOLN;
          IF DMODE[SAVECNT] = ' ' THEN
            BEGIN
              DMODE[SAVECNT] := '4';
              SAVECNT := SAVECNT + 1;
              DMODE[SAVECNT] := '0';
              SAVECNT := SAVECNT + 1;
              DMODE[SAVECNT] := '0';
              SAVECNT := SAVECNT + 1;
              DMODE[SAVECNT] := ' ';
              COUNTER := SAVECNT + 1;
            END;
          DMODE[COUNTER] := ';';
          REPEAT
            WRITE('SWITCH NORMAL OR SWITCH FOLLOW MODE (SN/SF): ');
            READLN(DMODE[COUNTER+1],DMODE[COUNTER+2]);
          UNTIL (DMODE[COUNTER+1] = 'S')
              AND ((DMODE[COUNTER+2] = 'N')
                    OR (DMODE[COUNTER+2] = 'F'));
          DMODE[COUNTER+3] := CHR(13);
        END;
    BUSOUT(DIGITIZER,ERFLAG,DMODE);
END;


PROCEDURE DIGERROR(VAR FINISH:BOOLEAN; VAR ERFLAG:CHAR);

VAR
  RESPONSE : ARRAY[1..10] OF CHAR;
  NUMBER, LENGTH : INTEGER;

BEGIN
  BUSOUT(DIGITIZER,ERFLAG,OUTERR);
  BUSIN(DIGITIZER,ERFLAG,RESPONSE);
  CASE RESPONSE[1] OF
    '1' :  (* INSTR NOT RECOGNIZED *)
                  WRITELN('DIG SAYS ''SAY WHAT'' ');
    '2' :  (* WRONG NUM PARAMS *)
                  WRITELN('WRONG NUM PARAMS');
    '3' :  (* BAD PARAMS REC *)
                  WRITELN('BAD PARAMS REC');
    '4' :  (* LABEL TOO LONG *)
```

```
                    WRITELN('LABEL TOO LONG');
      '6' :  (* OUT-OF-BOUNDS *)
             BEGIN
               WRITELN('OUT-OF-BOUNDS');
               BUSOUT(DIGITIZER,ERFLAG,DMODE);
             END;
      '7' :  (* LOW SIGNAL *)  WRITELN('LOW SIGNAL');
    END;
    FINISH := FALSE;
  END;

  PROCEDURE KEYFUNCTION(VAR FINISH:BOOLEAN; VAR ERFLAG:CHAR);

  VAR
    RESPONSE : ARRAY[1..10] OF CHAR;
    NUMBER, LENGTH : INTEGER;

  BEGIN
    BUSOUT(DIGITIZER,ERFLAG,OUTKEY);
    BUSIN(DIGITIZER,ERFLAG,RESPONSE);
    IF (RESPONSE[2] < '0') OR (RESPONSE[2] > '9')
     THEN LENGTH := 1
     ELSE
       IF (RESPONSE[3] < '0') OR (RESPONSE[3] > '9')
        THEN LENGTH := 2
        ELSE LENGTH := 3;
    WHATNUM(RESPONSE,LENGTH,NUMBER);
    FINISH := TRUE;
    CASE NUMBER OF
      1 :  (* Fa KEY ENTERED *)  BEGIN
                                   SAMPLING(ERFLAG);
                                   FINISH := FALSE;
                                 END;
      2 :  (* Fb KEY ENTERED *)  BEGIN
                                   KEYNUMBER := 2;
                                 END;
      4 :  (* Fc KEY ENTERED *)  BEGIN
                                   KEYNUMBER := 3;
                                 END;
      8 :  (* Fd KEY ENTERED *)  BEGIN
                                   KEYNUMBER := 4;
                                 END;
      16 :  (* Fe KEY ENTERED *) BEGIN
                                   KEYNUMBER := 5;
                                 END;
      32 :  (* PREFIX Fa KEY ENTERED *)  BEGIN
                                            KEYNUMBER := 6;
                                          END;
      64 :  (* PREFIX Fb KEY ENTERED *)  BEGIN
                                            KEYNUMBER := 7;
                                          END;
      128 :  (* PREFIX Fc KEY ENTERED *)  BEGIN
                                             KEYNUMBER := 8;
                                           END;
```

```
      256 :  (* PREFIX Fd KEY ENTERED *)  BEGIN
                                             KEYNUMBER := 9;
                                           END;
      512 :  (* PREFIX Fe KEY ENTERED *)  BEGIN
                                             KEYNUMBER := 10;
                                           END;
    END;
    BUSOUT(DIGITIZER,ERFLAG,CLRKEY);
  END;



  BEGIN (* GET_POINT *)
    IF KEYNUMBER < 0 THEN SAMPLING(ERFLAG);
    IF ERFLAG <> 'E' THEN
      BEGIN
        FINISH := FALSE;
        REPEAT
          BUSOUT(DIGITIZER,ERFLAG,STATUS);
          BUSIN(DIGITIZER,ERFLAG,RESPONSE);
          IF (RESPONSE[2] < '0') OR (RESPONSE[2] > '9')
           THEN LENGTH := 1
           ELSE
             IF (RESPONSE[3] < '0') OR (RESPONSE[3] > '9')
              THEN LENGTH := 2
              ELSE LENGTH := 3;
          WHATNUM(RESPONSE,LENGTH,NUMBER);
          IF TSTBIT(NUMBER,7) THEN KEYFUNCTION(FINISH,ERFLAG)
            ELSE
              IF TSTBIT(NUMBER,5)
                THEN DIGERROR(FINISH,ERFLAG)
                ELSE
                  IF TSTBIT(NUMBER,2) THEN
                    BEGIN
                      BUSOUT(DIGITIZER,ERFLAG,OUTPNT);
                      BUSIN(DIGITIZER,ERFLAG,DATA_LINE);
                      FINISH := TRUE;
                    END;
        UNTIL (ERFLAG = 'E') OR FINISH;
      END;
  END;


  MODEND.
```

Appendix B

This appendix contains the source code for the digitized line drawing input and output programs.

This is the source code for DIGITIZE and is contained in file DIGITIZE.SRC.

```
PROGRAM DIGITIZE;
(*  WRITTEN BY JOSEPH E. ROCK, JR.                                    *)
(*   DATE WRITTEN :  17 JULY 1983                                    *)
(*   PURPOSE :  THIS PROGRAM TAKES POINTS FROM THE DIGITIZER         *)
(*      AND PLACES THEM IN A USER SPECIFIED FILE.                    *)

TYPE
  LINE = ARRAY[1..40] OF CHAR;

VAR
  DATA:LINE;
  ERROR_FLAG:CHAR;
  COUNTER:INTEGER;
  F : TEXT;
  FILENAME : STRING;
  IO:INTEGER;
  SIGNAL:CHAR;
  ERFLAG:CHAR;
  KEYNUMBER: INTEGER;
  PEN: CHAR;
  FIRST_TIME : BOOLEAN;

EXTERNAL PROCEDURE GET_POINT(VAR DATA_LINE:LINE;
                      VAR ERFLAG:CHAR; VAR KEYNUMBER:INTEGER);
EXTERNAL FUNCTION PREPARE:CHAR;
PROCEDURE SETUP_FILE;

BEGIN
  WRITELN;
  WRITELN('THIS PROGRAM WILL WRITE OVER ANY EXISTING ',
              'FILE WITH THE SAME NAME');
  WRITE('ENTER FILENAME FOR DATA POINTS (EX: A:DATA1.DAT) :');
  READLN(FILENAME);
  ASSIGN(F,FILENAME);
  REWRITE(F);
  IF IORESULT = 255 THEN
    BEGIN
      WRITELN('ERROR IN CREATING FILE.. PROGRAM WILL TERMINATE');
      ERROR_FLAG := 'E';
    END
   ELSE
     BEGIN
       WRITELN('FILE OPENED SUCCESSFULLY');
       ERROR_FLAG := ' ';
       WRITELN;
       WRITELN('CONTROL KEYS ON DIGITIZER ARE:');
```

```
        WRITELN('  Fa = RESET DIGITIZER MODE');
        WRITELN('  Fb = CLOSE CURRENT FILE AND ASK FOR MORE');
        WRITELN('  Fc = PEN DOWN');
        WRITELN('  ANY OTHER KEY = PEN UP');
      END;
END;

BEGIN (* MAIN PROGRAM *)
  FIRST_TIME := TRUE;
  WRITELN('PROGRAM TO PUT DIGITIZED POINTS IN A FILE');
  ERROR_FLAG := PREPARE;
  REPEAT
    ERROR_FLAG := ' ';
    SETUP_FILE;
    PEN := 'U';
    SIGNAL := 'N';
    IF ERROR_FLAG <> 'E' THEN
      BEGIN
        WRITELN;
        WRITELN('NOW BEGIN TAKING POINTS');
        REPEAT
          REPEAT
            KEYNUMBER := 0;
            IF FIRST_TIME THEN KEYNUMBER := -1;
            GET_POINT(DATA,ERROR_FLAG,KEYNUMBER);
            FIRST_TIME := FALSE;
            IF KEYNUMBER <> 0 THEN
              IF KEYNUMBER = 3 THEN PEN := 'D'
                ELSE IF KEYNUMBER = 2 THEN
                    BEGIN
                      CLOSE(F,IO);
                      IF IO = 255
                       THEN WRITELN('ERROR IN CLOSING ',
                                    'FILE: ',FILENAME);
                      WRITE('MORE TO DIGITIZE (Y/N)? ');
                      READ(SIGNAL);
                      IF SIGNAL <> 'Y' THEN ERROR_FLAG := 'E'
                       ELSE
                         BEGIN  SETUP_FILE; PEN := 'U'; END;
                    END
                  ELSE PEN := 'U';
          UNTIL (KEYNUMBER = 0) OR (ERROR_FLAG = 'E');
          IF ERROR_FLAG <> 'E'  THEN
            BEGIN
              COUNTER := 1;
              WRITE(F,PEN,' ');
              REPEAT
                IF DATA[COUNTER] = ',' THEN DATA[COUNTER] := ' ';
                WRITE(F,DATA[COUNTER]);
                COUNTER := COUNTER + 1;
              UNTIL DATA[COUNTER] = CHR(13);
              WRITELN(F);
            END;
        UNTIL ERROR_FLAG = 'E';
      END;
```

```
  UNTIL ERROR_FLAG = 'E';
  WRITELN; WRITELN('ALL DONE .......');
END.
```

This is the source listing for the program PLOTFILE.

```
PROGRAM PLOTFILE;
(* WRITTEN BY JOSEPH E. ROCK, JR.                                    *)
(*  DATE WRITTEN :  17 JULY 1983                                     *)
(*  PURPOSE :  THIS PROGRAM PLOTS THE POINTS CONTAINED IN A          *)
(*     USER SPECIFIED FILE ON THE PLOTTER.  THIS PROGRAM ALSO*)
(*     SCALES AND TRANSLATES THE POINTS BEFORE PLOTTING THEM.*)

TYPE
  LINE = ARRAY[1..20] OF CHAR;

VAR
  ACKLINE,NUMLINE:LINE;
  PEN,ERROR_FLAG:CHAR;
  COUNTER:INTEGER;
  F : TEXT;
  FILENAME : STRING;
  IO:INTEGER;
  PLOTFRST,PLOTLINE:LINE;
  TEMP:REAL;
  SIGNAL,LINE_TYPE:CHAR;
  XYDATA:ARRAY[1..2] OF INTEGER;
  SCALE:REAL;
  TRANS:ARRAY[1..2] OF INTEGER;
  PLACE,LOOP:INTEGER;
  FIRST,FINISH:BOOLEAN;
  ERFLAG:CHAR;
  LENGTH:INTEGER;

EXTERNAL PROCEDURE PORTIN;
EXTERNAL PROCEDURE CHAROT(VAR OUTPUT:CHAR; VAR ERFLAG:CHAR);
EXTERNAL PROCEDURE CHARIN(VAR SIGNAL:CHAR; VAR ERFLAG:CHAR);
EXTERNAL PROCEDURE LINOUT(VAR CHARAC:LINE; VAR ERFLAG:CHAR);
EXTERNAL PROCEDURE WHATNUM(VAR CHARAC:LINE;
              VAR LENGTH,NUMB:INTEGER);
EXTERNAL PROCEDURE WHATCHAR(X:INTEGER; VAR LINEIN:LINE;
              VAR CNTR,LENGTH:INTEGER);

PROCEDURE PLOTIN;
BEGIN
  WRITE('ENTER PLOTTER LINE TYPE (0-8): ');
  READLN(LINE_TYPE);
  PLOTFRST[14] := LINE_TYPE;
  WRITELN;
  WRITE('ENTER SCALING FACTOR (DIG:PLOT)-  1:');
  READLN(SCALE);
  WRITE('ENTER PLOTTER TRANSLATION (X Y): ');
  READLN(TRANS[1],TRANS[2]);
```

```
    WRITELN;
    PLOTFRST[1] := ' ';
    PLOTFRST[2] := ';';
    PLOTFRST[3] := ':';
    PLOTFRST[4] := 'I';
    PLOTFRST[5] := 'O';
    PLOTFRST[6] := 'D';
    PLOTFRST[7] := ' ';
    PLOTFRST[8] := 'O';
    PLOTFRST[9] := ' ';
    PLOTFRST[10] := 'H';
    PLOTFRST[11] := 'A';
    PLOTFRST[12] := 'O';
    PLOTFRST[13] := 'L';
    PLOTFRST[15] := ' ';
    PLOTFRST[16] := CHR(13);
    PLOTFRST[17] := CHR(10);
    PLOTFRST[18] := '}';
    PLOTLINE[1] := ' ';
    PLOTLINE[2] := ';';
    PLOTLINE[3] := ':';
    PLOTLINE[4] := ' ';
    PLOTLINE[5] := '}';
    NUMLINE[2] := ' ';
    NUMLINE[7] := ',';
    NUMLINE[12] := ' ';
    NUMLINE[13] := '}';
    ACKLINE[1] := CHR(13);
    ACKLINE[2] := CHR(10);
    ACKLINE[3] := '}';
END;

PROCEDURE SETUP_FILE;

BEGIN
  WRITELN;
  WRITE('ENTER FILENAME OF DATA POINTS (EX: A:DATA1.DAT) : ');
  READLN(FILENAME);
  ASSIGN(F,FILENAME);
  RESET(F);
  IF IORESULT = 255 THEN
    BEGIN
      WRITELN(' UNABLE TO OPEN FILE... PROGRAM WILL TERMINATE');
      ERROR_FLAG := 'E';
    END
   ELSE
    BEGIN
      WRITELN('FILE OPENED SUCCESSFULLY');
      ERROR_FLAG := ' ';
    END;
END;


BEGIN (* MAIN PROGRAM *)
```

```
    WRITELN('PROGRAM TO PLOT DIGITIZED POINTS FROM A FILE');
    ERROR_FLAG := ' ';
    PORTIN;
    SETUP_FILE;
    PLOTIN;
    WRITELN;
    FIRST := TRUE;
    LINOUT(PLOTFRST,ERROR_FLAG);
    CHARIN(SIGNAL,ERROR_FLAG);
    REPEAT
      LINOUT(PLOTLINE,ERROR_FLAG);
      COUNTER := 16;
      REPEAT
        READLN(F,PEN,XYDATA[1],XYDATA[2]);
        NUMLINE[1] := PEN;
        IF FIRST THEN BEGIN NUMLINE[1] := 'U'; FIRST := FALSE END;
        FOR LOOP := 1 TO 2 DO
          BEGIN
            TEMP := XYDATA[LOOP] * SCALE;
            XYDATA[LOOP] := ROUND(TEMP);
            XYDATA[LOOP] := XYDATA[LOOP] + TRANS[LOOP];
            LENGTH := 4;
            IF LOOP = 1 THEN PLACE := 3  ELSE PLACE := 8;
            WHATCHAR(XYDATA[LOOP],NUMLINE,PLACE,LENGTH);
          END;
        LINOUT(NUMLINE,ERROR_FLAG);
        COUNTER := COUNTER + 12;
        FINISH := EOF(F);
      UNTIL (COUNTER >= 254) OR (ERROR_FLAG = 'E') OR FINISH;
      LINOUT(ACKLINE,ERROR_FLAG);
      CHARIN(SIGNAL,ERROR_FLAG);
    UNTIL (ERROR_FLAG = 'E') OR FINISH;
    LINOUT(PLOTFRST,ERROR_FLAG);
    CHARIN(SIGNAL,ERROR_FLAG);
    WRITELN('ALL DONE .......');
  END.
```

This is the source for the program DIGPLOT.

```
PROGRAM DIGPLOT;
(* WRITTEN BY JOSEPH E. ROCK, JR.                                    *)
(*  DATE WRITTEN :  25 MARCH 1983                                    *)
(*  PURPOSE :  THIS PROGRAM TAKES POINTS FROM THE DIGITIZER          *)
(*     AND PLOTS THEM ON THE PLOTTER AT THE SAME TIME.               *)
(*     THERE IS AN ALLOWANCE FOR SCALING AND TRANSLATING THEM*)
(*     POINTS AS THEY ARE BEING PLOTTED.                             *)

TYPE
  LINE = ARRAY[1..40] OF CHAR;

VAR
  NUMARRAY,DATA:LINE;
  ERROR_FLAG:CHAR;
  NUM_POINT:INTEGER;
  COUNTER:INTEGER;
  F : TEXT;
  FILENAME : STRING;
  IO:INTEGER;
  PLOTLINE:LINE;
  PLOTCNT:INTEGER;
  TEMP:REAL;
  SIGNAL,LINE_TYPE:CHAR;
  XYDATA:ARRAY[1..2] OF INTEGER;
  SCALE:REAL;
  TRANS:ARRAY[1..2] OF INTEGER;
  LOOP,CNTNUM:INTEGER;
  ERFLAG:CHAR;
  LENGTH,I:INTEGER;
  KEYNUMBER: INTEGER;
  FIRST_TIME: BLLOEAN;

EXTERNAL PROCEDURE BUSIN(DEVICE:INTEGER; VAR ERFLAG:CHAR;
            VAR INLINE:LINE);
EXTERNAL PROCEDURE BUSOUT(DEVICE:INTEGER; VAR ERFLAG:CHAR;
            VAR OUTLINE:LINE);
EXTERNAL FUNCTION BUSINT:CHAR;
EXTERNAL PROCEDURE PORTIN;
EXTERNAL PROCEDURE CHAROT(VAR OUTPUT:CHAR; VAR ERFLAG:CHAR);
EXTERNAL PROCEDURE CHARIN(VAR SIGNAL:CHAR; VAR ERFLAG:CHAR);
EXTERNAL PROCEDURE LINOUT(VAR CHARAC:LINE; VAR ERFLAG:CHAR);
EXTERNAL PROCEDURE GET_POINT(VAR DATA_LINE:LINE;
            VAR ERFLAG:CHAR; VAR KEYNUMBER:INTEGER);
EXTERNAL FUNCTION PREPARE:CHAR;
EXTERNAL PROCEDURE WHATNUM(VAR CHARAC:LINE; VAR LENGTH,NUMB:INTEGER
EXTERNAL PROCEDURE WHATCHAR(X:INTEGER; VAR LINEIN:LINE; VAR CNTR,LE
```

```
PROCEDURE PLOTIN;
BEGIN
  WRITE('ENTER PLOTTER LINE TYPE (0-8): ');
  READLN(LINE_TYPE);
  PLOTLINE[14] := LINE_TYPE;
  WRITELN;
  WRITE('ENTER SCALING FACTOR (DIG:PLOT)-  1:');
  READLN(SCALE);
  WRITE('ENTER PLOTTER TRANSLATION (X Y): ');
  READLN(TRANS[1],TRANS[2]);
  WRITELN;
  WRITELN('NOW BEGIN TAKING POINTS');
  PLOTLINE[11] := 'U';
  PLOTLINE[1] := ';';
  PLOTLINE[2] := ':';
  PLOTLINE[3] := 'I';
  PLOTLINE[4] := ' ';
  PLOTLINE[5] := '0';
  PLOTLINE[6] := 'D';
  PLOTLINE[7] := ' ';
  PLOTLINE[8] := '0';
  PLOTLINE[9] := ' ';
  PLOTLINE[10] := 'A';
  PLOTLINE[11] := 'U';
  PLOTLINE[12] := ' ';
  PLOTLINE[13] := 'L';
  PLOTLINE[15] := ' ';
END;


BEGIN (* MAIN PROGRAM *)
  WRITELN('PROGRAM TO PLOT DIGITIZED POINTS AS THEY ',
            'ARE DIGITIZED');
  ERROR_FLAG := PREPARE;
  FIRST_TIME := TRUE;
  PORTIN;
  PLOTIN;
  SAMPLING(ERROR_FLAG);
  WRITELN;
  WRITELN('NOW BEGIN TAKING POINTS');
  REPEAT
    ERFLAG := ' ';
    REPEAT
      KEYNUMBER := 0;
      IF FIRST_TIME THEN KEYNUMBER := -1;
      GET_POINT(DATA,ERROR_FLAG,KEYNUMBER);
      IF KEYNUMBER <> 0 THEN
        IF KEYNUMBER = 3 THEN PLOTLINE[11] := 'D'
          ELSE IF KEYNUMBER = 2 THEN PLOTIN
            ELSE PLOTLINE[11] := 'U';
    UNTIL (KEYNUMBER = 0) OR (ERROR_FLAG = 'E');
    IF ERROR_FLAG <> 'E' THEN
      BEGIN
        PLOTCNT := 16;
        COUNTER := 1;
```

```
            FOR LOOP := 1 TO 2 DO
              BEGIN
                CNTNUM := 1;
                REPEAT
                  NUMARRAY[CNTNUM] := DATA[COUNTER];
                  CNTNUM := CNTNUM + 1;
                  COUNTER := COUNTER + 1;
                UNTIL DATA[COUNTER] = ',';
                COUNTER := COUNTER + 1;
                CNTNUM := CNTNUM - 1;
                WHATNUM(NUMARRAY,CNTNUM,XYDATA[LOOP]);
                TEMP := XYDATA[LOOP] * SCALE;
                XYDATA[LOOP] := ROUND(TEMP);
                XYDATA[LOOP] := XYDATA[LOOP] + TRANS[LOOP];
                LENGTH := 4;
                WHATCHAR(XYDATA[LOOP],PLOTLINE,PLOTCNT,LENGTH);
                PLOTCNT := PLOTCNT + 1;
                PLOTLINE[PLOTCNT] := ',';
                PLOTCNT := PLOTCNT + 1;
              END;
            PLOTCNT := PLOTCNT - 1;
            PLOTLINE[PLOTCNT] := ' ';
            PLOTCNT := PLOTCNT + 1;
            PLOTLINE[PLOTCNT] := CHR(13);
            PLOTCNT := PLOTCNT + 1;
            PLOTLINE[PLOTCNT] := CHR(10);
            PLOTCNT := PLOTCNT + 1;
            PLOTLINE[PLOTCNT] := '}';
            LINOUT(PLOTLINE,ERROR_FLAG);
            CHARIN(SIGNAL,ERFLAG);
        END;
      UNTIL ERROR_FLAG = 'E';
      WRITELN('ALL DONE .......');
    END.
```

This is the source code for LABELS.

```
PROGRAM LABELS;
(* PROGRAM TO PRINT CHARACTER STRINGS ON THE PLOTTER    *)
TYPE
  LINE = ARRAY[1..256] OF CHAR;

VAR
  LINEOUT : LINE;
  X, Y :INTEGER;
  COUNTER : INTEGER;
  INLINE : LINE;
  LINECNT : INTEGER;
  ERFLAG : CHAR;
  DELTAX,DELTAY : INTEGER;
  ANSWER : CHAR;
  CNTLP,LENGTH, TIMES : INTEGER;


EXTERNAL PROCEDURE PORTIN;
EXTERNAL PROCEDURE CHAROT(VAR OUTPUT:CHAR; VAR ERFLAG:CHAR);

PROCEDURE CONVERT(VAR X,Y:INTEGER; VAR LINEIN:LINE;
          VAR CNTR:INTEGER);
VAR
  TEMP1:INTEGER;
  TEMP:INTEGER;
  LOOPCNT:INTEGER;


PROCEDURE WHATCHAR(VAR NUMBER:INTEGER; VAR NUMCH:CHAR);
BEGIN
  CASE NUMBER OF
  0 : NUMCH := '0';
  1 : NUMCH := '1';
  2 : NUMCH := '2';
  3 : NUMCH := '3';
  4 : NUMCH := '4';
  5 : NUMCH := '5';
  6 : NUMCH := '6';
  7 : NUMCH := '7';
  8 : NUMCH := '8';
  9 : NUMCH := '9';
  ELSE NUMCH := 'E';
  END;
END;

BEGIN
  TEMP := X DIV 1000 ;
```

```
    WHATCHAR(TEMP,LINEIN[CNTR]);
    CNTR := CNTR + 1;
    TEMP1 := TEMP * 1000;
    TEMP := (X - TEMP1) DIV 100;
    WHATCHAR(TEMP,LINEIN[CNTR]);
    CNTR := CNTR + 1;
    TEMP1 := TEMP1 + TEMP * 100;
    TEMP := (X - TEMP1) DIV 10;
    WHATCHAR(TEMP,LINEIN[CNTR]);
    CNTR := CNTR + 1;
    TEMP := X - TEMP1 - TEMP * 10;
    WHATCHAR(TEMP,LINEIN[CNTR]);
    CNTR := CNTR + 1;
    LINEIN[CNTR] := ',';
    CNTR := CNTR + 1;
  (* NOW FOR THE Y VALUE   *)
    TEMP := Y DIV 1000;
    WHATCHAR(TEMP,LINEIN[CNTR]);
    CNTR := CNTR + 1;
    TEMP1 := TEMP * 1000;
    TEMP := (Y - TEMP1) DIV 100;
    WHATCHAR(TEMP,LINEIN[CNTR]);
    CNTR := CNTR + 1;
    TEMP1 := TEMP1 + TEMP * 100;
    TEMP := (Y - TEMP1) DIV 10;
    WHATCHAR(TEMP,LINEIN[CNTR]);
    CNTR := CNTR + 1;
    TEMP := Y - TEMP1 - TEMP * 10;
    WHATCHAR(TEMP,LINEIN[CNTR]);
    CNTR := CNTR + 1;
    LINEIN[CNTR] := ' ';
    CNTR := CNTR + 1;
  END;


  BEGIN (* MAIN PROGRAM *)
    WRITELN('PROGRAM TO PRINT STRINGS ON THE PLOTTER');
    PORTIN;
    REPEAT
      WRITE('BEGINNING COORDINATES (''X Y''):');
      READLN(X,Y);
      LINEOUT[1] := ';';
      LINEOUT[2] := ':';
      LINEOUT[3] := 'I';
      LINEOUT[4] := ' ';
      LINEOUT[5] := '0';
      LINEOUT[6] := 'D';
      LINEOUT[7] := ' ';
      LINEOUT[8] := '2';
      LINEOUT[9] := ' ';
      LINEOUT[10] := 'A';
      LINEOUT[11] := 'U';
      LINECNT := 12;
      CONVERT(X,Y,LINEOUT,LINECNT);
      WRITE('ENTER COMMAND STRING:');
```

```
        REPEAT
          READ(LINEOUT[LINECNT]);
          LINECNT := LINECNT + 1;
        UNTIL EOLN;
        LENGTH := LINECNT - 1;
        WRITE('REPEAT HOW MANY TIMES: ');
        READLN(TIMES);
        IF TIMES <> 1 THEN
          BEGIN
            WRITE('ENTER X & Y INCREMENTS (1 3): ');
            READLN(DELTAX,DELTAY);
          END;
        CNTLP := 1;
        FOR CNTLP := 1 TO TIMES DO
          BEGIN
            FOR COUNTER := 1 TO LENGTH DO
              CHAROT(LINEOUT[COUNTER],ERFLAG);
              X := X + DELTAX;
              Y := Y + DELTAY;
              LINECNT := 12;
              CONVERT(X,Y,LINEOUT,LINECNT);
          END;
        WRITE('ANOTHER LINE (Y/N)? ');
        READ(ANSWER);
        WRITELN;
      UNTIL ANSWER = 'N';
    END.
```

This is the source for the number to character and character to number routines that were developed.

```
MODULE CONVERT;

TYPE LINE = ARRAY[1..20] OF CHAR;

PROCEDURE WHATCHAR(X:INTEGER; VAR LINEIN:LINE;
            VAR CNTR,LENGTH:INTEGER);
VAR
  NUMBER:INTEGER;
  TEMP:INTEGER;
  LOOPCNT: INTEGER;


BEGIN
  TEMP := CNTR;
  CNTR := CNTR + LENGTH - 1;
  FOR LOOPCNT := CNTR DOWNTO TEMP DO
    BEGIN
      TEMP := X DIV 10;
      NUMBER := X - (TEMP * 10);
      CASE NUMBER OF
        1 : LINEIN[LOOPCNT] := '1';
        2 : LINEIN[LOOPCNT] := '2';
        3 : LINEIN[LOOPCNT] := '3';
        4 : LINEIN[LOOPCNT] := '4';
        5 : LINEIN[LOOPCNT] := '5';
        6 : LINEIN[LOOPCNT] := '6';
        7 : LINEIN[LOOPCNT] := '7';
        8 : LINEIN[LOOPCNT] := '8';
        9 : LINEIN[LOOPCNT] := '9';
        ELSE LINEIN[LOOPCNT] := '0';
      END;
      X := TEMP;
    END;
END;


PROCEDURE WHATNUM(VAR CHARAC:LINE; VAR LENGTH,NUMB:INTEGER);
VAR
  CNTR:INTEGER;
  MULT:INTEGER;
  DIG:INTEGER;
BEGIN
  NUMB:=0;
  MULT:=1;
  FOR CNTR := LENGTH DOWNTO 1 DO
  BEGIN
```

```
        CASE CHARAC[CNTR] OF
        '0' : DIG := 0;
        '1' : DIG := 1;
        '2' : DIG := 2;
        '3' : DIG := 3;
        '4' : DIG := 4;
        '5' : DIG := 5;
        '6' : DIG := 6;
        '7' : DIG := 7;
        '8' : DIG := 8;
        '9' : DIG := 9;
        END;
        NUMB := NUMB + DIG*MULT;
        MULT := MULT*10;
      END;
    END;

    MODEND.
```

Appendix C

This appendix contains the source code for the chain coding program and the area error computation program.

This is the source for the chain coding program CODER.

```
PROGRAM CODER;

(*                                                              *)
(*WRITTEN BY: JOSEPH E. ROCK, JR                                 *)
(* DATE: 23 JULY 1983                                           *)
(* PURPOSE:  THE FUNCTION OF THIS PROGRAM IS TO CONVERT A       *)
(*   SET OF X-Y COORDINATES INTO THE FIRST LEVEL CHAIN CODE     *)
(*   FOR THE DRAWING. THE OUTPUT FILE WILL CONTAIN THE          *)
(*   STARTING X-Y COORDINATE AND THE GRIDSIZE USED IN           *)
(*   COMPUTING THE CHAIN CODE FOR EVERY CONTINUOUS LINE         *)
(*   SEGMENT.  THE OUTPUT FILE FORMAT IS :                      *)
(*     PEN X-COORDINATE Y-COORDINATE CHAIN CODE                 *)
(*A -1 CHAIN CODE INDICATES THAT THE POINT IS THE FIRST         *)
(*     OR LAST IN THE  LINE SEGMENT                             *)

TYPE
  RELARRAY = ARRAY[1..2] OF REAL;
  INTARRAY = ARRAY[1..3,1..2] OF INTEGER;

VAR
  TEMP : REAL;
  XCODE,YCODE : INTEGER;
  LEVEL,GRIDSIZE : INTEGER;
  DELTAXY : RELARRAY;
  POINT : INTARRAY;
  FIRST, FINISH : BOOLEAN;
  PEN : CHAR;
  FILENAME : STRING;
  F, G : TEXT;
  RESULT, NUMBER : INTEGER;
  CODEB, DELTA : INTEGER;

EXTERNAL PROCEDURE WHERE(VAR F:TEXT;DELTA:INTEGER;
     VAR DELTAXY:RELARRAY;VAR POINT:INTARRAY;VAR PEN:CHAR);


PROCEDURE STARTUP;
(* INTERACTIVE PARAMETER INITIALIZATION SECTION *)
  BEGIN
    WRITELN('PROGRAM TO COMPUTE SINGLE RING CHAIN CODE');
    WRITE('ENTER INPUT FILE NAME (A:DATA.DAT) : ');
    READLN(FILENAME);
    ASSIGN(F,FILENAME);
    WRITE('ENTER OUTPUT FILE NAME (A:DATA.CD1) : ');
    READLN(FILENAME);
    ASSIGN(G,FILENAME);
    RESET(F);
    REWRITE(G);
```

```
    WRITE('ENTER GRIDSIZE TO BE USED (INTEGER > 0) : ');
    READLN(GRIDSIZE);
    WRITE('ENTER LEVEL OF CODE (INTEGER > 0) : ');
    READLN(LEVEL);
  END;

PROCEDURE COMPCODE;

BEGIN
  TEMP := (DELTAXY[1] + DELTA) / GRIDSIZE;
  XCODE := ROUND(TEMP);
  TEMP := (DELTAXY[2] + DELTA) / GRIDSIZE;
  YCODE := ROUND(TEMP);
  POINT[1,1] := POINT[1,1] + (XCODE - LEVEL) * GRIDSIZE;
  POINT[1,2] := POINT[1,2] + (YCODE - LEVEL) * GRIDSIZE;
  IF XCODE >= YCODE
   THEN CODEB := XCODE + YCODE
   ELSE CODEB := (LEVEL * 8) - XCODE - YCODE;
  IF (CODEB < (3 * LEVEL))
   THEN CODEB := CODEB + LEVEL * 5
   ELSE CODEB := CODEB - LEVEL * 3;
  IF FIRST THEN BEGIN PEN := 'U'; FIRST := FALSE; END
           ELSE PEN := 'D';
END;



BEGIN (* MAIN PROGRAM *)
  STARTUP;
  DELTA := GRIDSIZE * LEVEL;
  FINISH := FALSE;
  WHILE NOT FINISH DO
    BEGIN
      FIRST := FALSE;
      NUMBER := 0;
      REPEAT
        POINT[2] := POINT[3];
        READLN(F,PEN,POINT[3,1],POINT[3,2]);
        NUMBER := NUMBER + 1;
        FINISH := EOF(F);
      UNTIL (PEN = 'D') OR FINISH;
      IF NOT FINISH
       THEN
         BEGIN
           IF NUMBER < 2 THEN
             BEGIN
               POINT[2] := POINT[3];
               READLN(F,PEN,POINT[3,1],POINT[3,2]);
             END;
           POINT[1] := POINT[2];
           WRITELN(G,'U ',POINT[1,1],' ',POINT[1,2],' -1 '
                        ,GRIDSIZE,' ',LEVEL);
           REPEAT
             WHERE(F,DELTA,DELTAXY,POINT,PEN);
             CASE PEN OF
```

```
                'E' :   BEGIN
                          FINISH := TRUE;
              WRITELN(G,'D ',POINT[2,1],' ',POINT[2,2],' -1')
                        END;
                'D' :   BEGIN
                          COMPCODE;
        WRITELN(G,PEN,' ',POINT[1,1],' ',POINT[1,2],' ',CODEB);
                        END;
                'U' :   BEGIN
                          FIRST := TRUE;
                 WRITELN(G,'D ',POINT[2,1],' ',POINT[2,2],' -1')
                        END
              END;
            UNTIL FIRST OR FINISH;
          END;
    END; (* END OF WHILE NOT FINISH LOOP *)
  CLOSE(G,RESULT);
  WRITELN('COMPUTATIONS COMPLETE');
END.
```

This is the source for the WHERE routine used in CODER.

```
MODULE WHERE;
(* THIS PROCEDURE IS ENTERED WITH THE DIGITIZED POINTS FILE *)
(* NAME, THE GRID CENTER, THE LAST TWO DIGITIZED POINTS     *)
(* READ IN AND THE GRIDSIZE BEING USED.  THE PROCEDURE      *)
(* RETURNS WITH THE POINT WHERE THE DIGITIZED LINE INTERSECTS*)
(* THE SQUARE WITH SIDE LENGTH GRIDSIZE AND CENTER THE       *)
(* GIVEN GRID CENTER.  THE PROCEDURE ALSO RETURNS WITH A    *)
(* PEN VALUE INDICATING:                                    *)
(*       U  =  END OF CURRENT LINE SEGMENT FOUND            *)
(*       D  =  NORMAL GRID INTERSECTION                     *)
(*       E  =  END OF FILE FOUND                            *)
(*  IF PEN <> D THEN WHERE RETURNS THE DELTAXY FOR          *)
(* THE LAST POINT                                           *)

TYPE INTARRAY = ARRAY[1..3,1..2] OF INTEGER;
     RELARRAY = ARRAY[1..2] OF REAL;

VAR FINISH : EXTERNAL BOOLEAN;

PROCEDURE WHERE(VAR F:TEXT;DELTA:INTEGER;
     VAR DELTAXY:RELARRAY;VAR POINT:INTARRAY;VAR PEN:CHAR);

VAR  DISTX,DISTY : INTEGER;(* F = DIGITIZED POINT FILE        *)
     TEMP : REAL;           (* DELTA = LENGTH OF SIDE OF GRID  *)
     SIGNX : INTEGER;       (* DELTAXY = INTERSECTION DELTAS   *)
                            (* POINT[1] IS GRID CENTER         *)
                           (* POINT[2] & [3] ARE LAST LINE PTS *)

PROCEDURE THEREST;
BEGIN
  IF POINT[3,1] = POINT[2,1]
   THEN (* VERTICAL LINE.. INTERSECT = (0, +- DELTA)  *)
     IF POINT[1,2] <= POINT[3,2] THEN DELTAXY[2] := DELTA
      ELSE DELTAXY[2] := -1 * DELTA
   ELSE
     BEGIN (* CALCULATE INTERSECTION POINT *)
       TEMP := (POINT[3,2] - POINT[2,2]) /
                    (POINT[3,1] - POINT[2,1]);
       IF POINT[3,1] >= POINT[2,1] THEN SIGNX := 1
          ELSE SIGNX := -1;
       TEMP := TEMP * (SIGNX * DELTA - POINT[2,1] + POINT[1,1]);
       DELTAXY[2] := TEMP + POINT[2,2] - POINT[1,2];
       IF ABS(DELTAXY[2]) > DELTA THEN
         IF DELTAXY[2] > 0 THEN DELTAXY[2] := DELTA
           ELSE DELTAXY[2] := -1 * DELTA;
     END;
  IF (POINT[3,2] = POINT[2,2]) OR (ABS(DELTAXY[2]) < DELTA)
```

```
    THEN (* INTERSECT LEFT OR RIGHT SIDE *)
      IF POINT[1,1] <= POINT[3,1] THEN DELTAXY[1] := DELTA
       ELSE DELTAXY[1] := -1 * DELTA
    ELSE
      BEGIN (* CALCULATE X INTERCEPT OF SQUARE GRID *)
        TEMP := (POINT[3,1] - POINT[2,1]) /
                        (POINT[3,2] - POINT[2,2]);
        TEMP := TEMP * (DELTAXY[2] - POINT[2,2] + POINT[1,2]);
        DELTAXY[1] := TEMP + POINT[2,1] - POINT[1,1];
      END;
END;


BEGIN (* MAIN PROCEDURE *)
  IF PEN <> 'U' THEN
    BEGIN
      DISTX := ABS(POINT[3,1] - POINT[1,1]) - DELTA;
      DISTY := ABS(POINT[3,2] - POINT[1,2]) - DELTA;
      WHILE (DISTX < 0) AND (DISTY < 0)
       AND (PEN <> 'U') AND NOT FINISH DO
        BEGIN (* REPEAT UNTIL CROSS GRID OR END LINE SEG *)
          POINT[2] := POINT[3];
          READLN(F,PEN,POINT[3,1],POINT[3,2]);
          IF PEN <> 'U' THEN
            BEGIN
              DISTX := ABS(POINT[3,1] - POINT[1,1]) - DELTA;
              DISTY := ABS(POINT[3,2] - POINT[1,2]) - DELTA;
            END;
        FINISH := EOF(F);
        END;
      IF ((DISTX >= 0) OR (DISTY >= 0)) AND (PEN <> 'U')
       THEN THEREST
       ELSE
         BEGIN
           IF FINISH THEN
            BEGIN PEN := 'E';
              DELTAXY[1] := POINT[3,1] - POINT[1,1];
              DELTAXY[2] := POINT[3,2] - POINT[1,2]
            END
           ELSE
            BEGIN
              DELTAXY[1] := POINT[2,1] - POINT[1,1];
              DELTAXY[2] := POINT[2,2] - POINT[1,2]
            END;
         END;
    END; (* END OF FIRST THEN CLAUSE *)
END;

MODEND.
```

This is the source for the area error computation program ERROR.

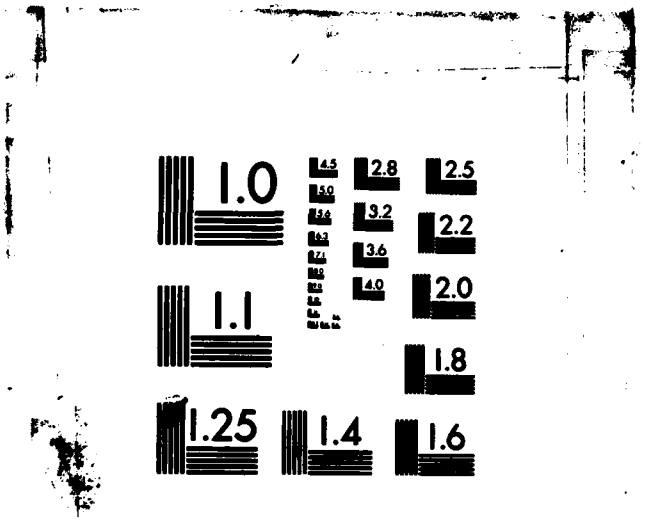
```
PROGRAM ERROR;

(*  WRITTEN BY JOSEPH E. ROCK, JR.                                    *)
(*    DATE: 3 SEPT 1983                                               *)
(*    THE PURPOSE OF THIS PROGRAM IS TO COMPUTE THE AREA              *)
(*  ERROR BETWEEN A DIGITIZED LINE AND THE CODED VERSION OF           *)
(*  THAT LINE.  THE PROGRAM ASKS FOR THE FILENAME OF THE              *)
(*  DATA OF THE DIGITIZED LINE AND THE FILENAME OF THE DATA           *)
(*  FOR THE CODED VERSION OF THAT LINE.  THE FILES ARE                *)
(*  ASSUMED TO BE IN THE FORMAT THAT I HAVE USED FOR                  *)
(*  DIGITIZED DATA AND CODED LINE DATA RESPECTIVELY.                  *)

TYPE  RELARRAY = ARRAY[1..2,1..2] OF REAL;

VAR
  FILENAME : STRING;
  C,D : TEXT;
  XVAL,YVAL,NUMCODE,NUMPTS,I,NOCODE,NOPTS : INTEGER;
  PEN : CHAR;
  CODED,DIG : ARRAY[1..60,1..2] OF REAL;
  LSTPTS : RELARRAY;
  SEGINTR : ARRAY[1..2] OF REAL;
  FINISH,ERFLAG : BOOLEAN;
  AREA : REAL;
  LONGCODE,LONGDIG : REAL;
  LASTXD,LASTYD,LASTXC,LASTYC : INTEGER;


PROCEDURE CLOSELOOP;
 VAR CROSS : BOOLEAN;


FUNCTION DIGAHEAD:BOOLEAN;
 VAR SIGNX,SIGNY:INTEGER;

BEGIN
  SIGNX := ROUND(CODED[NUMCODE - 1,1] - CODED[NUMCODE,1]);
  SIGNY := ROUND(CODED[NUMCODE - 1,2] - CODED[NUMCODE,2]);
  IF SIGNX <> 0
   THEN IF SIGNX > 0 THEN SIGNX := 1  ELSE SIGNX := -1;
  IF SIGNY <> 0
   THEN IF SIGNY > 0 THEN SIGNY := 1  ELSE SIGNY := -1;
  IF (SIGNX * (CODED[NUMCODE,1] - DIG[NUMPTS,1]) > 0) OR
     (SIGNY * (CODED[NUMCODE,2] - DIG[NUMPTS,2]) > 0)
   THEN DIGAHEAD := TRUE
   ELSE DIGAHEAD := FALSE;
```

AD-A138 413 A MICROCOMPUTER BASED SYSTEM FOR ANALYSIS OF LINE DRAWING QUANTIZATION TECHNIQUES(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. J E ROCK

2/2

UNCLASSIFIED 09 DEC 83 AFIT/GE/EE/83D-77 F/G 9/2 NL

END



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
END;

PROCEDURE LASTCODE;
BEGIN
  REPEAT
    READLN(D,PEN,XVAL,YVAL);
    LENGTHD;
    NUMPTS := NUMPTS + 1;
    DIG[NUMPTS,1] := XVAL;
    DIG[NUMPTS,2] := YVAL;
  UNTIL EOF(D) OR (PEN <> 'D');
  FINISH := TRUE;
END;

PROCEDURE LASTDIG;
BEGIN
  REPEAT
    READLN(C,PEN,XVAL,YVAL);
    LENGTHC;
    NUMCODE := NUMCODE + 1;
    DIG[NUMCODE,1] := XVAL;
    DIG[NUMCODE,2] := YVAL;
  UNTIL EOF(C) OR (PEN <> 'D');
  FINISH := TRUE
END;

PROCEDURE INTERSECT;
VAR
  SAME : BOOLEAN;
  SIGNX,SIGNY : INTEGER;
  SEG1,SEG2 : RELARRAY;

PROCEDURE FINDPT;
VAR SLOPE1,SLOPE2,B1,B2,DIST1,DIST2 : REAL;
    XAXIS,YAXIS : INTEGER;

PROCEDURE FINDY;
VAR SLOPE,B : REAL;

BEGIN
  IF SEG1[2,XAXIS] <> SEG1[1,XAXIS]
   THEN
     BEGIN
       SLOPE := (SEG1[2,YAXIS] - SEG1[1,YAXIS]) /
                    (SEG1[2,XAXIS] - SEG1[1,XAXIS]);
       B := SEG1[1,YAXIS] - SLOPE * SEG1[1,XAXIS];
       SEGINTR[YAXIS] := SEGINTR[XAXIS] * SLOPE + B;
     END
   ELSE
     BEGIN
       SLOPE := (SEG2[2,YAXIS] - SEG2[1,YAXIS]) /
                    (SEG2[2,XAXIS] - SEG2[1,XAXIS]);
       B := SEG2[1,YAXIS] - SLOPE * SEG2[1,XAXIS];
       SEGINTR[YAXIS] := SEGINTR[XAXIS] * SLOPE + B;
     END;
```

```
END;

PROCEDURE FINDIT;
VAR SLOPE1,SLOPE2,B1,B2 : REAL;

BEGIN
  SLOPE1 := (SEG1[2,YAXIS] - SEG1[1,YAXIS]) /
                (SEG1[2,XAXIS] - SEG1[1,XAXIS]);
  B1 := SEG1[1,YAXIS] - SLOPE1 * SEG1[1,XAXIS];
  SLOPE2 := (SEG2[2,YAXIS] - SEG2[1,YAXIS]) /
                (SEG2[2,XAXIS] - SEG2[1,XAXIS]);
  B2 := SEG2[1,YAXIS] - SLOPE2 * SEG2[1,XAXIS];
  IF SLOPE1 = SLOPE2  THEN CROSS := FALSE
   ELSE
     BEGIN
       SEGINTR[XAXIS] := (B2 - B1) / (SLOPE1 - SLOPE2);
       SEGINTR[YAXIS] := SLOPE1 * SEGINTR[XAXIS] + B1;
      END
END;

PROCEDURE CHECKPNT;
BEGIN
  SIGNX := ROUND(CODED[NOCODE - 1,1] - CODED[NOCODE,1]);
  IF SIGNX <> 0 THEN IF SIGNX > 0
   THEN SIGNX := 1  ELSE SIGNX := -1
   ELSE BEGIN SIGNX := ROUND(DIG[NOPTS - 1,1] - DIG[NOPTS,1]);
          IF SIGNX >= 0 THEN SIGNX := 1 ELSE SIGNX := -1; END;
  SIGNY := ROUND(CODED[NOCODE - 1,2] - CODED[NOCODE,2]);
  IF SIGNY <> 0 THEN IF SIGNY > 0
    THEN SIGNY := 1  ELSE SIGNY := -1
   ELSE BEGIN SIGNY := ROUND(DIG[NOPTS - 1,2] - DIG[NOPTS,2]);
          IF SIGNY >= 0 THEN SIGNY := 1 ELSE SIGNY := -1; END;
  IF (SIGNX * (SEGINTR[1] - CODED[NOCODE - 1,1]) <= 0)
   AND (SIGNX * (CODED[NOCODE,1] - SEGINTR[1]) <= 0)
   AND (SIGNY * (SEGINTR[2] - CODED[NOCODE - 1,2]) <= 0)
   AND (SIGNY * (CODED[NOCODE,2] - SEGINTR[2]) <= 0)
   AND (SIGNX * (SEGINTR[1] - DIG[NOPTS - 1,1]) <= 0)
   AND (SIGNX * (DIG[NOPTS,1] - SEGINTR[1]) <= 0)
   AND (SIGNY * (SEGINTR[2] - DIG[NOPTS - 1,2]) <= 0)
   AND (SIGNY * (DIG[NOPTS,2] - SEGINTR[2]) <= 0)
     THEN CROSS := TRUE
     ELSE CROSS := FALSE;
 END;

BEGIN (* FINDPT *)
  CROSS := TRUE;
  IF ABS(SEG1[2,1] - SEG1[1,1]) >= ABS(SEG1[2,2] - SEG1[1,2])
   THEN BEGIN XAXIS := 1; YAXIS := 2 END
   ELSE BEGIN XAXIS := 2; YAXIS := 1 END;
  IF ABS(SEG1[2,1] - SEG1[1,1]) = ABS(SEG1[2,2] - SEG1[1,2])
   THEN
    IF ABS(SEG2[2,1] - SEG2[1,1]) >= ABS(SEG2[2,2] - SEG2[1,2])
     THEN BEGIN XAXIS := 1; YAXIS := 2 END
     ELSE BEGIN XAXIS := 2; YAXIS := 1 END;
  IF SEG1[2,XAXIS] = SEG1[1,XAXIS]
```

```
    THEN
      BEGIN
        SEGINTR[XAXIS] := SEG1[1,XAXIS];
        IF SEG2[2,XAXIS] = SEG2[1,XAXIS]
         THEN
           IF SEG1[1,XAXIS] = SEG2[1,XAXIS]
            THEN
              IF ABS(SEG1[1,YAXIS] - SEG1[2,YAXIS])
                   <= ABS(SEG1[1,YAXIS] - SEG2[2,YAXIS])
                THEN SEGINTR[YAXIS] := SEG1[2,YAXIS]
                ELSE SEGINTR[YAXIS] := SEG2[2,YAXIS]
            ELSE CROSS := FALSE
         ELSE FINDY
      END
    ELSE
      IF SEG2[2,XAXIS] = SEG2[1,XAXIS]
       THEN
         BEGIN
           SEGINTR[XAXIS] := SEG2[1,XAXIS];
           FINDY
         END
       ELSE FINDIT;
  IF CROSS THEN CHECKPNT;
END;


BEGIN (* INTERSECT *)
  IF NUMCODE <= 3  THEN NOCODE := 1
    ELSE NOCODE := NUMCODE - 2;
  REPEAT
    SEG1[1,1] := CODED[NOCODE,1];
    SEG1[1,2] := CODED[NOCODE,2];
    NOCODE := NOCODE + 1;
    SEG1[2,1] := CODED[NOCODE,1];
    SEG1[2,2] := CODED[NOCODE,2];
    IF NUMPTS <= 3 THEN NOPTS := 1 ELSE NOPTS := NUMPTS - 2;
    IF (NOCODE <= 2) AND (NUMPTS >= 3) THEN NOPTS := 2;
    REPEAT
      SEG2[1,1] := DIG[NOPTS,1];
      SEG2[1,2] := DIG[NOPTS,2];
      NOPTS := NOPTS + 1;
      SEG2[2,1] := DIG[NOPTS,1];
      SEG2[2,2] := DIG[NOPTS,2];
      FINDPT;
    UNTIL CROSS OR (NOPTS >= NUMPTS);
  UNTIL CROSS OR (NOCODE >= NUMCODE);
END;



BEGIN (* CLOSELOOP *)
  IF (CODED[1,1] = LSTPTS[2,1]) AND (CODED[1,2] = LSTPTS[2,2])
   THEN NUMCODE := 1;
  CODED[NUMCODE,1] := LSTPTS[2,1];
  CODED[NUMCODE,2] := LSTPTS[2,2];
```

```
    IF (DIG[1,1] = LSTPTS[1,1]) AND (DIG[1,2] = LSTPTS[1,2])
     THEN NUMPTS := 1;
    DIG[NUMPTS,1] := LSTPTS[1,1];
    DIG[NUMPTS,2] := LSTPTS[1,2];
    REPEAT
      IF DIGAHEAD
       THEN
         BEGIN
           READLN(C,PEN,XVAL,YVAL);
           LENGTHC;
           NUMCODE := NUMCODE + 1;
           CODED[NUMCODE,1] := XVAL;
           CODED[NUMCODE,2] := YVAL;
           IF EOF(C) OR (PEN <> 'D') THEN LASTCODE
         END
       ELSE
         BEGIN
           READLN(D,PEN,XVAL,YVAL);
           LENGTHD;
           NUMPTS := NUMPTS + 1;
           DIG[NUMPTS,1] := XVAL;
           DIG[NUMPTS,2] := YVAL;
           IF EOF(D) OR (PEN <> 'D') THEN LASTDIG
         END;
      IF (NUMPTS >= 60) OR (NUMCODE >= 60)
        THEN BEGIN
              WRITELN('--- ERROR - TOO MANY POINTS BETWEEN',
                'INTERSECTIONS - ERROR');
              ERFLAG := TRUE;
      IF NOT FINISH THEN INTERSECT;
    UNTIL CROSS OR FINISH OR ERFLAG;
    IF NOT FINISH AND NOT ERFLAG
     THEN
       BEGIN
         LSTPTS[2,1] := CODED[NUMCODE,1];
         LSTPTS[2,2] := CODED[NUMCODE,2];
         LSTPTS[1,1] := DIG[NUMPTS,1];
         LSTPTS[1,2] := DIG[NUMPTS,2];
         CODED[NUMCODE,1] := SEGINTR[1];
         CODED[NUMCODE,2] := SEGINTR[2];
         DIG[NUMPTS,1] := SEGINTR[1];
         DIG[NUMPTS,2] := SEGINTR[2];
       END;
END;

PROCEDURE GROUND;
VAR SUBAREA,XORIGIN : REAL;
    I : INTEGER;

BEGIN
  XORIGIN := CODED[1,1];
  FOR I := 2 TO NUMCODE DO
    IF CODED[I,1] < XORIGIN  THEN XORIGIN := CODED[I,1];
  FOR I := (NUMPTS - 1) DOWNTO 2 DO
    IF DIG[I,1] < XORIGIN  THEN XORIGIN := DIG[I,1];
```

```
    SUBAREA := 0;
    FOR I := 2 TO NUMCODE DO
      SUBAREA := SUBAREA + (CODED[I,2] - CODED[I-1,2]) *
             ((CODED[I-1,1] + CODED[I,1])/2 - XORIGIN);
    FOR I := NUMPTS DOWNTO 2 DO
      SUBAREA := SUBAREA + (DIG[I-1,2] - DIG[I,2]) *
             ((DIG[I-1,1] + DIG[I,1])/2 - XORIGIN);
      AREA := AREA + ABS(SUBAREA);
  END;

  PROCEDURE LENGTHD;
  BEGIN
    LONGDIG := LONGDIG + SQRT(SQR(XVAL - LASTXD) +
          SQR(YVAL - LASTYD));
    LASTXD := XVAL;
    LASTYD := YVAL;
  END;

  PROCEDURE LENGTHC;
  BEGIN
    LONGCODE := LONGCODE + SQRT(SQR(XVAL - LASTXC) +
          SQR(YVAL - LASTYC));
    LASTXC := XVAL;
    LASTYC := YVAL;
  END;


  BEGIN (* MAIN PROGRAM *)
    AREA := 0.0;
    LONGCODE := 0.0;
    LONGDIG := 0.0;
    ERFLAG := FALSE;
    WRITELN;
    REPEAT
      WRITE('ENTER DIGITIZED DATA FILENAME: ');
      READLN(FILENAME);
      ASSIGN(D,FILENAME);
      RESET(D);
    UNTIL IORESULT <> 255;
    REPEAT
      WRITE('ENTER CODED DATA FILENAME: ');
      READLN(FILENAME);
      ASSIGN(C,FILENAME);
      RESET(C);
    UNTIL IORESULT <> 255;
    READLN(C,PEN,XVAL,YVAL);
    CODED[1,1] := XVAL;
    CODED[1,2] := YVAL;
    LASTXC := XVAL;
    LASTYC := YVAL;
    READLN(C,PEN,XVAL,YVAL);
    LSTPTS[2,1] := XVAL;
    LSTPTS[2,2] := YVAL;
    LENGTHC;
    REPEAT
```

```
      READLN(D,PEN,XVAL,YVAL);
      DIG[1,1] := LSTPTS[1,1];
      DIG[1,2] := LSTPTS[1,2];
      LSTPTS[1,1] := XVAL;
      LSTPTS[1,2] := YVAL;
    UNTIL PEN = 'D';
    LASTXD := ROUND(DIG[1,1]);
    LASTYD := ROUND(DIG[1,2]);
    LENGTHD;
    REPEAT
      NUMCODE := 2;
      NUMPTS := 2;
      CLOSELOOP;
      IF NOT ERFLAG THEN
        BEGIN
          GROUND;
          CODED[1,1] := SEGINTR[1];
          CODED[1,2] := SEGINTR[2];
          DIG[1,1] := SEGINTR[1];
          DIG[1,2] := SEGINTR[2];
        END;
    UNTIL FINISH OR ERFLAG;
    WRITELN(' AREA = ',AREA);
    WRITELN(' LENGTH OF CODED LINE = ',LONGCODE);
    WRITELN(' LENGTH OF DIGITIZED LINE = ',LONGDIG);
  WRITELN('THAT''S ALL FOLKS..... ');
  END.
```

Appendix D

This appendix is a user's manual for the system of software and hardware described in this thesis.

User's Manual

A list of the available programs and a brief statement of their purpose is shown below.

DIGITIZE : digitize a line drawing on the digitizer and store the data in a disk file

PLOTFILE : plot the data stored in a disk file

DIGPLOT : echo the points being digitized on the digitizer to the plotter

LABELS : provide direct control of the plotter

CODER : compute the single ring chain code of a line drawing described by the series of coordinates stored in a file and output the code to a disk file

ERROR : compute the area error between the line drawings described by the data in two disk files, also computes the length of the line in each disk file

A more complete guide to using each of these programs follows.

DIGITIZE

To run the digitizer program enter

A0>DIGITIZE (CR)

The program will promt the user for the filename to be used and the sampling parameters for the digitizer. The special function keys of the digitizer are defined as :

Fa : reset digitizer sampling parameters

Fb : close current file and ask user for next filename or end of program

Fc : indicate pen down

Fd through prefix Fe : indicate pen up

PLOTFILE

The program to plot a set of data points is invoked by:

A0>PLOTFILE (CR)

The program will prompt the user for the filename and the parameters for the plotter. The plotter parameters are :

Line Type (integer 0-8) : The available line types are:

L0

L1

L2

L3

L4

L5

L6

L7

L8

Scaling Factor : The x and y coordinates of the input file are multiplied by the scaling factor, added to their respective translation factors, and output to the plotter.

The scaling factor is a real number and must be entered as x.y (a value of .2 would be entered as 0.2). The coordinates in the files normally have units of 0.001 in. The plotter uses units of 0.005 in for all numbers. Therefore, a 1 in = 1 in scale for a digitized drawing would be

X * (in / 1000) * (200 / in) = X * 0.2 (scaling factor = 0.2)

The translation number is in units of 0.005 in (200 = 1 in).

Translation factor : 200 = 1 in (described above)

DIGPLOT

The digitizer to plotter program is invoked by

A0>DIGPLOT (CR)

The digitizer to plotter program prompts the user for all of the input it requires. The input required has been described in the instruction for DIGITIZE and PLOTFILE. The program is exited by entering a CNTRL-C from the keyboard.

The special function keys are defined as:

Fa : reset digitizer sampling parameters

Fb : reset plotter parameters

Fc : indicate pen down

Fd - prefix Fe : indicate pen up

LABELS

The plotter direct control program is invoked by

A0>LABELS (CR)

The program prompts the user for all input. If a mistake is made in entering the command line then the only way to correct it is to specify a repetition number of zero. The x and y increments are integer numbers with unit of 0.005 in. A summary of the plotter commands follows:

O - set origin: the current pen location becomes the new origin

D - pen down: puts the pen down at the current location

U - pen up: immediately picks the pen up

H - home: moves the pen to the home location (lower left corner) and defines that location as the new origin

A - absolute plotting mode: all coordinates will be plotted with respect to the currently defined origin

R - relative plotting mode: all coordinates will be plotted with respect to the point plotted immediately prior to the point being currently plotted

Ln - set line type: define line type as n (see line type definition above)

Srhbsss_ - symbol plotting: plot ASCII character string sss with rotation r and height h ('_' indicates end of character string, b is a space) (r is an integer 1 - 4 and the rotation is: rotation = (r-1)*90 degrees, 0 degrees is straight right and the rotation angle is positive clockwise) (h is an integer 1 - 5 which corresponds to hieghts of: 1 = 0.07", 2 = 0.14", 3 = 0.28", 4 = 0.56", and 5 = 1.12")

T - self-test routine: perform self-test program

x,y - move pen to x,y: move the pen to x,y with respect to the origin or previous point (A vs R) with the pen up or down (U vs D) (x and y are integers sent as ASCII character strings)

A more complete description of the commands can be found in the plotter's instruction manual[4].

CODER

The single ring chain coding program is invoked by

AO>CODER (CR)

The program will prompt the user for all the input it needs. The user must enter the input digitized line drawing filename, the output filename, and the level and gridsize of the code. The units of the gridsize (assuming a line drawing input using DIGITIZE) factor are 0.001 in (1000 means a gridsize of 1 in).

ERROR

The area error computation program is invoked by

AO>ERROR (CR)

The program will prompt the user for all of the input it requires. The user must input the filenames of the two lines to be used for the calculation. The program will notify the user if the limits of the program are exceeded.

Vita

Joseph Edward Rock, Jr was born on 30 April 1959 in Clearfield, Pennsylvania. He graduated from high school at St. Clairsville, Ohio in 1977. In 1981 he graduated with a Bachelor of Science of Electrical Engineering from Ohio University was assigned to the School of Engineering, Air Force Institute of Technology.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GE/EE/83D-77	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Air Force Institute of Technology; 6b. OFFICE SYMBOL (If applicable): AFIT	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Wright-Patterson AFB OH 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO.; PROJECT NO.; TASK NO.; WORK UNIT NO.
11. TITLE (Include Security Classification): A Microcomputer Based System for Analysis of Line Drawing Quantization Techniques (U)	
12. PERSONAL AUTHOR(S): Joseph E. Rock, Jr., 1Lt, USAF	
13a. TYPE OF REPORT: MS Thesis; 13b. TIME COVERED: FROM Jun 81 TO Dec 83; 14. DATE OF REPORT (Yr., Mo., Day): 1983 Dec 09; 15. PAGE COUNT: 93	
16. SUPPLEMENTARY NOTATION: Feb 84 — Approved for public release: IAW AFR 190-17 — LYNN E. WOLAVER, Dean for Research and Professional Development, Air Force Institute of Technology (ATC), Wright-Patterson AFB OH 45433	
17. COSATI CODES — FIELD; GROUP; SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Line Drawing; Chain Codes; Computer Graphics

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This paper documents the design and implementation of a system for analysis of line drawing quantization techniques on a microcomputer system. The system provides software tools for the input, output, and analysis of line drawings and their quantized representations. The system uses a digitized version of the original line drawing as the basis for all performance calculations.

A program to compute the single ring chain code of the input digitized line drawing was implemented. The performance of the chain coded versions of the line drawings could then be calculated. This project developed all the necessary tools for the analysis of any line drawing quantization technique that can be implemented on a microcomputer system.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL: Kenneth G. Castor, Major, USAF	22b. TELEPHONE NUMBER (Include Area Code): 513-255-5533; 22c. OFFICE SYMBOL: ENG

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE


FIL